Optimizing the Workload Scheduling in MEC-Assisted Intelligent Transportation Systems

Ebrahim Sarkhouh

A Thesis
In
The Department of
Computer Science & Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy (Computer Science)
Concordia University
Montréal, Québec, Canada

September 2022

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: **Mr. Ebrahim Sarkhouh**

Entitled: **Optimizing the Workload Scheduling in MEC-Assisted Intelligent Transportation Systems**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

_____ Chair

Prof. Georges Kaddoum _____ External Examiner

Dr. Mustafa Mehmet Ali _____ External to Program

Dr. Emad Shihab _____ Examiner

Prof. Lata Narayanan _____ Examiner

Prof. Chadi Assi _____ Supervisor

Approved _____
Chair of Department or Graduate Program Director

_____ 2021 _____

Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

ABSTRACT

Optimizing the Workload Scheduling in MEC-Assisted Intelligent Transportation Systems

Ebrahim Sarkhouh, Ph.D.
Concordia University, 2022

Autonomous driving (AD) is rising as an efficient solution to a wide range of transportation problems. With all the capabilities utilized (sensors, 5G communication technologies, computation units), intelligent vehicles can interact with the surroundings and cooperate in instantaneously maneuvering safely and effectively. Incorporating a central agent that supports this on-the-road interaction represents a critical enabling idea that will elevate the Cooperative Autonomous Driving (CAD) performance. Multi-access Edge Computing (MEC) recently attracted a considerable focus, specifically in vehicular networks, as it provides a reliable and online response to service demands arriving from vehicles. In the context of CAD, MEC can reduce the usage of wireless communications, orchestrate the activities on the road and provide massive computation capabilities to the vehicles. In this dissertation, we investigate the potential of MEC in the context of supporting autonomous driving and managing the radio and computation resources available. We propose adequate solutions for various problems that MEC should continuously resolve as an essential component of a complete intelligent transportation system.

First, we examine the capability of MEC by formulating the problem of scheduling vehicular computational tasks over the resources as an optimization problem and solve it via integer linear programming (ILP) and Lagrangian relaxation. We prove the complexity of the problem, and thus we develop a scalable solution that reaches near-optimal solutions and around 90% speedup compared with branch-and-cut.

Second, to consolidate our work veracity, we tighten the system capacity by limiting the wireless communication resources. Also, we propose a system model that harvests the computational resources available on the vehicles' onboard units via a fog computing scheme and utilizes them along with the infrastructure edge resources.

The system aims to jointly allocate the radio and computational resources to maximize the number of admitted tasks. We provide a formal definition of the problem as multi-stage scheduling and, due to its complexity, propose a Dantzing-Wolfe decomposition method to solve the problem. We compare the performance of the proposed method with CPLEX and show that the solution is only 20% far from the optimal solution while achieving 94% speedup.

After demonstrating the merit of deploying/utilizing edge servers in a vehicular network, the third contribution particularizes more the system model to an AD environment by applying two significant modifications. First, we accurately represent an AD scenario by modeling the computational load as long-term processes that continuously receive data from multiple sources, process them together, and inform multiple destinations with decisions supporting cooperative autonomous driving applications. Such processes work as assistants to on-the-road activities such as changing lanes, taking turns, or establishing/maintaining platoons. Second, we adopt a sophisticated, more suitable metric that quantifies the freshness of the information received called Age of Information (AoI). We aim to minimize AoI of the information continuously received in the destinations. The problem turned to be NP-hard. We propose a novel Benders decomposition technique that divides the problem into several subproblems and one integer master problem. We developed a scalable solution for each of these problems and compared the overall method with the optimal solution. The method proposed showed high scalability and efficiency in terms of the objective and computation time.

We conclude with a discussion on the outcomes of this thesis and the directions we intend to take in our future work.

Acknowledgments

I want to thank my supervisor for his guidance and support during these fantastic four years of my life. Although he gave me hard times, it was obvious to me that he was doing that to motivate me as he believed in me. I want to thank you sincerely, Prof. Assi, for everything. Thank you for giving me the chance to work with you and for all the support you provided.

Also, I want to thank Dr. Dariush Ebrahimi. He was there all along. Thank you, doctor, for your productive discussions. Thank you for your guidance and the knowledge you shared. I have to thank also Dr. Sanaa Sharafeddine, Dr. Ribal Atallah, and Dr. Maurice Khabbaz for their guidance, their support and their contributions with me in the projects of this thesis.

I also want to express my sincere appreciation to the committee members, Prof Lata Narayanan, Dr. Emad Shihab, and Dr. Mustafa Mehmet Ali, for their guidance during my Ph.D. study. Thank you very much for your constructive comments and your valuable feedback. I would also like to thank Dr. Georges Kaddoum for his acceptance to be my external examiner.

I want to mention two mentors here, who made Ebrahim what he is right now by showing him what research is about and how passion should be the motivation to work in anything. Prof. Maytham Safar, thank you so much for being a brother to me. Thank you for the faith you showed in my capabilities. Thank you for being beside me all the time during this journey. Dr. Khaled Mahdi, thank you for being a brother. Thank you for your guidance and support. Thank you both for the knowledge you provided. Thanks to both of you for teaching me how to be a researcher.

My most tremendous thanks is to my parents—two great teachers, Mr. Yousef Sarkhouh and Mrs. Anisah Sarkhouh, who taught me never to wait for someone to feed me knowledge. They taught me that I should always rely on myself. That success is not about grades or GPA. It is about the knowledge you gain. Thank you that you

were there for me during the hard and good times. You were the first to celebrate with me and the first to support me. Thank you for being the most incredible parents. Being my parents is the most lavish blessing I have ever had.

To my brothers, Anwer, Ahmed, and Osamah. To my sisters, Zainab and Zahra'. Thank you all for the support. Thank you all for the faith.

Finally, I want thank all my friends, especially those who were with me in the lab. Thank you, Sadegh, Hyame, Shirin, Ekram, and Elie. It was my great honor and pleasure that we shared this lab. We had great discussions back then and helped each other in work and life.

To my precious son and daughter, Yousef and Yara.
I love you 3000...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As an essential element of Intelligent Transportation Services (ITS), Autonomous and teleoperated Vehicles (AV) play a significant role in providing a safe yet effective service that overcomes a wide range of problems that we all face daily [5]. With the advanced sensors (e.g., radars, LIDARS, GPS, and motion sensors) [6], communication technologies (i.e., C-V2X) [7], and computing capabilities deployed on the On-Board Units (OBUs), AV constitutes an intelligent entity capable of perceiving various types of detailed information and interacting accordingly with its surrounding. The massive leap in Artificial Intelligence (AI) in vehicles allows them to see, navigate, and maneuver within the typical transportation systems. With the rise of the Internet of Things (IoT) service ecosystem as an enabling concept to intelligent cities, AV became part of a socio-technical environment of users and other competent entities that instantaneously cooperate to accomplish an on-the-road task (e.g., changing lanes) or to perform specific analytics on the condition of the transportation system [8]. Indeed, AVs are increasingly rising as an effective solution to many of our on-the-road daily problems. It represents a computational asset deployed with various AI applications. Computational demand can also harvest such an asset via a proper framework. Yet, establishing a direct connection to a moving vehicle in this dynamic environment is an intricate task. Although AVs went through a revolutionizing stage, most current vehicles lack sufficient computational capabilities.

1.1 Motivation

The lack of a central agent capable of interacting with the vehicles and establishing and managing the required tasks to be accomplished inflicts several issues obstructing the realization of fully reliable intelligent transportation services. First, while the computational resources depict the system's core, wireless communication resources

are more critical, considering its minimal capacity and the high demand for the required data transmissions. A central agent will mitigate this criticality by reducing the number of coinciding data transmissions. Second, as stated before, most of the vehicles currently do not have adequate computational capabilities. With computation resources deployed over the infrastructure, these vehicles can benefit from these resources by offloading their required computational task to these resources. Last, with distributed data analytics, decisions taken from a vehicle might not suit other decisions taken by other vehicles or IoT devices. Hence, the employment of what is called *Multi-access Edge Computing (MEC)* [9] paradigm is essential to ITS.

MEC in recent years has emerged as a performance/reliability shifting paradigm by bringing the computation processes from the cloud to the very edge of the network, thereby providing a unique opportunity for latency-critical services. In vehicular networks, this edge is the Road Side Unit (RSU) that interconnects with the vehicles and gives them an internet service and a Vehicle to Vehicle (V2V) connectivity. The RSU can receive a computational workload from the vehicles, process it at the co-located edge server, and send it back to the vehicles [10]. This workload can be requests to instantiate autonomous driving application instances or perform certain data analytics that requires a comprehensive view of the state of the road. RSUs with MEC can act as a controller that orchestrates the various activities on the road by utilizing the IoT devices and the vehicle's sensors to observe the overall road state and accordingly provide the required support/assistance to the vehicles. However, the installation of multiple highly-empowered cloudlets might incur considerable capital and operational expenditures. These expenses motivate the necessity of an alternative cost-minimal solution that accounts for both latency requirements and the tasks' admissibility. This level is where Vehicular Fog Computing (VeFC) [11] and the concept of the vehicle as a Resource (VaaR) [12] come into play.

Fog computing is a framework that facilitates the exploitation of computational resources available on the edge of the network. By utilizing idle computational resources at that level, we can expand fog computing beyond the infrastructure resources (MEC) to exploit any network edge node (i.e., vehicle). VeFC broaches the concept of VaaR by harvesting the vehicle's OBUs computational capabilities. Over a long time, if the vehicles dedicate their OBUs to their own vehicles' internal processing, their capabilities most probably will be underutilized. As these vehicles' OBUs possess, AI tools [13], other vehicles that lack such resources can improve their driving automation potentials through wireless computational tasks offloading. As stated earlier, the central MEC agent can monitor the available resources and control the utilization of these resources. For example, a vehicle can request from other vehicles, through the MEC agent, to sense their surrounding environments, analyze, and send back the results to the requester to support the creation of lifelike Augmented Reality objects that support both short-term and long-term navigation. Besides, IoT devices and pedestrians may also benefit from the computational capabilities available on OBUs. For example, IoT devices that require a road network status may provide vehicles with data or ask vehicles to execute some artificial intelligent tasks over some data and send results back to them.

1.2 Research Objectives

The main objective of this thesis is to provide solutions to various concerns that should be tackled to achieve the vision of complete intelligent transportation systems. Specifically, we suggested system models that assume the availability of computational resources right on the network's edge (i.e., RSU). We intend to allocate these computing units and the available wireless communication channels for the requests coming from the vehicle and other intelligent entities to accomplish workload over the

edge or other entities to optimize a particular metric. In addition to the conventional throughput metric, we propose to optimize what is called *Age of Information (AoI)* [14]. Mainly, we will try to answer the following research questions:

1. ***How can a vehicular network utilize an edge server deployed over an RSU?***

   We need to study the feasibility of deploying a massive computational resource right on the edge of the network. The system model is an RSU (the network edge) provided with an edge server and receives requests to accomplish computational tasks. A Vehicle-to-RSU communication takes place over a 5G wireless communication platform. We assume in this research question that such a communication technology provides sufficient bandwidth, making the scheduling mainly focused on the computational resources.

2. ***Is it efficient to jointly schedule the wireless and computational resources in vehicular edge computing?***

   Scheduling problems reserve a considerable part in the theory of computation. That's because they are usually challenging problems to solve (i.e., NP-Hard). This thesis tries to tackle multi-stage scheduling problems via decomposition techniques and empirically find the efficiency of deploying such solutions on an edge server to utilize the available resources.

3. ***How can an MEC agent orchestrate the utilization of the available computational resources in a fog-enabled vehicular network?***

   As with the stationary servers, computational capacity in vehicles is growing enough to consider it an asset to utilize. In this research question, we try to leverage these assets by making the edge server (deployed over the RSU)

establishes connections with the vehicle to receive computational tasks and send them to other vehicles to compute.

4. ***What is the role of information freshness in vehicular networks, and can it improve the quality of service?***

   AoI quantifies the information received in the destination by its recency. In our last contribution, we try to find how we can optimize such a metric and discusses the metric validity in vehicular networks.

5. ***How to make edge computing supports cooperative autonomous driving applications with a maximized information freshness?***

   This question is the central question that this thesis is trying to answer. Can MEC support/assist CAD applications by receiving data from multiple vehicles, analyzing it, and providing the vehicle with beneficial information? To answer it, we should carry along the answers to more than one question asked earlier and integrate their solutions to come up with this question response. MEC should apply an efficient scheduler to allocate wireless and computational resources for processes that continuously receive data from various entities and support the on-the-road with decisions taken while considering very fresh received information.

1.3 Contributions

In this section, we summarize our main contributions presented and discussed in this thesis:

1.3.1 Workload Scheduling in Vehicular Networks with Edge Cloud Capabilities

To support the development of 5G technologies, researchers are actively engaged in addressing the challenges accompanying the emerging 5G applications. Unquestionably, a rising technology gaining significant research attention is edge computing. Vehicular Edge Computing (VEC) brings data storage, and computing capabilities and hosts support applications that comprise emerging vehicular services that demand low-delay processing to the edge closer to the vehicles. This approach reduces response times and increases reliability, therefore achieving the holistic vision of the tactile Internet. In this context, this contribution considers a vehicular network with edge computing capabilities deployed at roadside units. It addresses the problem of workload offloading and scheduling of computation tasks on the computing resources available at the edge. The challenges here are the high mobility of the vehicles and the high sensitivity of the delay the computational tasks exhibit. We formulate a problem considering the computation resources and the latency requirements of the workload and prove the scheduling to be NP-Hard. Subsequently, efficient solutions based on Lagrangian relaxation are derived and presented. We evaluate the proposed methods numerically and show their closeness to the optimal solutions.

1.3.2 An Infrastructure-Assisted Workload Scheduling for Computational Resources Exploitation in Fog-Enabled Vehicular Network

Now that we have established and evaluated a comprehensive VEC system model, we want to expand the idea of utilizing the available computational resources available on the infrastructure to harvesting the resources available on the vehicles themselves.

The vehicle-as-a-resource is an emerging concept that allows the exploitation of the vehicles' computational resources to execute tasks offloaded by passengers, vehicles, or even internet-of-things devices. This contribution revolves around a scenario where a roadside unit located at the edge of a hierarchical multi-tier edge computing sub-network resorts to utilize idle vehicles' computational resources through a fog-enabled substructure, yielding a cost-effective computational tasks offloading solution. In this context, scheduling the offload of these tasks to the appropriate vehicles is a challenging problem that is subject to the interaction of major role-playing parameters. Among these parameters are the variability of vehicles' availability and their computational power, the individual tasks' weighted priorities and deadlines, the required computational power, and the required data to upload/download. This work proposes an infrastructure-assisted task scheduling scheme where the roadside unit receives computational tasks from different sources and schedule these tasks over computationally capable vehicles within the roadside unit's range. The aim is to maximize the weighted number of admitted tasks while considering the constraints mentioned above. Our system considers both the latency and throughput of tasks accomplishments by maximizing the weighted number of admitted tasks while at the same time respecting the tasks accompanied deadlines. Both radio and computational resources are part of the optimization problem. After proving the NP-hardness of the scheduling problem, we formulated the problem as a mixed-integer linear program. A Dantzig-Wolfe decomposition algorithm is proposed, which yields to a master program solvable by the Barrier algorithm and subproblems solved optimally with a polynomial-time dynamic programming approach. We conducted a thorough numerical analysis and simulations to verify and assert our approach's validity, correctness, and effectiveness compared to branch-and-bound and greedy algorithms.

1.3.3 Optimizing Information Freshness for MEC-enabled Cooperative Autonomous Driving

The previous contributions showed how MEC provides a plethora of computational services to reduce network latency and maximize the throughput. Here, we want to be more specific in the form of the workload offloaded to the edge server in an AD environment and the metric to optimize for this workload. Applications at the edge that apply analytics on the sensory data are indispensable for self-driving vehicles. We consider in this contribution a network that interconnects vehicles to an edge server at a roadside unit. Each vehicle extracts multiple information by sampling multiple processes and sends them to the corresponding edge application. To make timely decisions, "fresh" information needs to be offloaded, processed, and delivered back to vehicles; in this context, we adopt a recently proposed metric called Age of Information that measures the freshness of information. We seek to jointly schedule vehicles' transmission of information and schedule information processing at the edge to minimize the AoI of all processes. We mathematically formulate the problem and prove its NP-Hardness. We propose a logic-based Benders decomposition to divide the problem into a Master and several subproblems to overcome this hardness. Then, we present an exact polynomial-time solution for the subproblems, a scalable heuristic for the master, and devise a valid yet efficient Benders cut. We implement the system simulation on the well-known traffic simulator SUMO and compare the decomposition with CPLEX branch-and-cut; Although the problem is highly intricate, our method finds a near-optimal solution (maximum deviation is 7% from optimal solution) with a speedup that reaches 95%. We study the system performance by varying different system parameters.

1.4 Thesis Organization

This thesis structure follows: Chapter 2 is about the related work and a brief presentation of the background required. Chapter 3 introduces our contribution solving the scheduling problem of tasks offloaded to the edge server of an RSU. Chapter 4 discusses vehicular fog computing and the concept of vehicle-as-resource with jointly scheduling the radio and computational resources. We discuss the third contribution in chapter 5, where we model a system to support cooperative autonomous driving applications via edge computing. We conclude this thesis in chapter 6 and our future directions.

Chapter 2

Background and Related Work

This chapter briefly summarizes several notions, paradigms and concepts that we utilize through this thesis. We present the concepts of edge computing, fog computing, vehicular networks, and autonomous driving. Also, we highlight the recent related work of these concepts and this thesis contribution.

2.1 Background

2.1.1 Multi-Access Edge Computing

Multi-access edge computing is *a paradigm that deploys computation and data storage on the lowest level of a network hierarchy hence closer to the users' equipments* [9].

Even though it is a simple concept, MEC benefits are immense. It amplifies the performance of computational tasks offloading, data caching, and networking in terms of latency and reliability.

According to [15], and compared to Mobile Cloud Computing (MCC), edge computing has the following advantages:

1. Low latency: in terms of information propagation, the distance between an edge server and the users is in the scale of meters, while in the case of MCC, the information should pass the entire core network to reach the cloud servers, causing a considerable delay.

2. Power efficiency: IoT devices' primary concern is the short battery life. Providing such computation resources nearby gives these devices the choice to offload computational-intensive tasks to these resources reducing the amount of energy consumed while processing these tasks.

3. Context Awareness: As MEC servers are on closer points to the users, they can

track real-time information like their location, behaviors, and environment.

4. Privacy/Security Enhancement: As MCC servers have central natures, they are vulnerable to attack due to their high information concentration. Since MCC separates management and ownership of user data, this might cause private data leakage as well. Edge servers overcome these issues by being proximate and distributed over the network edges.

As mentioned in [16], Nokia and IBM installed the first edge computing servers in 2013. European Telecommunications Standards Institute (ETSI) established MEC in 2014 and formed the MEC Industry Specification Group (ISG) that standardizes the deployment of MEC over Radio Access Networks (RAN). In general, MEC technologies are now under the deployment stage. Nevertheless, the reports clearly state that we will soon see edge servers deployed on a large scale.

In vehicular networks, *MEC servers are deployed over the RSUs in order to provide computational capabilities for the vehicles.* Such a deployment is what is called **vehicular edge computing** paradigm [17].

2.1.1.1 Fog Computing

Fog computing is *an expansion to the utilization of the infrastructure's computational resources, as in MEC, to monitoring, managing and offloading tasks to idle resources available on the users' equipments* [11].

The inflation of computational and data transfer demand requires an economical solution that provides a sufficient constancy in the availability of the resources. As the MEC servers have a specific capacity, researchers tried to acquire more resources by deploying edge servers in a hierarchical fashion, allowing the lower-level servers to offload some of the computation to the higher-level servers, increasing the admission rate increasing the service speed. With this arrangement, lower-tier cloudlets are

allowed to issue task migration requests to upper cloudlet tiers. Analytical studies were conducted in [18] and [19] demonstrating the superiority of Hierarchical MECs (H-MECs)' over typical flat MECs in terms of delay and task admissibility. However, installation of multiple proprietary cloudlets and their organization in H-MECs incur considerable capital and operational expenditures (*e.g.* [20]). These expenditures motivate the necessity of an alternative cost-minimal solution that accounts for both latency requirements and the tasks' admissibility. However, such a solution is not practical and increases the overall infrastructure establishment cost. Here is where the paradigm of fog computing takes place to lower this cost by enabling the utilization of idle resources that belong to the users equipments.

2.1.2 Vehicular Networks

Vehicular network is *the communication platform that interconnects the vehicles, the on-the-road IoT devices and RSUs in order to enhance the vehicles' sensing capabilities and intelligence* [2]. In this subsection, we will discuss its early history and its current available technologies.

2.1.2.1 Early History

The following is a summary of the vehicular networks' early history as mentioned in [2]. In the 1980s and 1990's most of the vehicles were already carrying radio-set as standard. At this time, the purpose of this radio set was to receive some weather updates or some urgent news. Given its purpose, it was reasonable to have such simple unidirectional communication technology. In the 1990s, Philips invented dedicated short-range communications (DSRC) to provide limited telecommunication and internet services for vehicles. The range was 5 meters, and the bandwidth was limited. Around the year 2000, the world realized the first actual vehicular ad-hoc

network (VANET). By increasing the bandwidth assigned to the DSRC protocol suite, it was possible to increase the range of the communication to 500 meters and increase the rate up to 26 Mb/s. That allowed researchers to build a system that provides safety services such as collision avoidance, ramp access control, fog, ice, and obstacle alerts. Since in each 500 meters road segment, most probably, there is a vehicle, it was not necessary to make the cars communicate only through the infrastructure; hence they allowed the vehicle to communicate in an ad-hoc fashion. In parallel to all that, cellular networks were going through tremendous evolution and passing four different generations of technologies where the communication bit rate has reached around 100 Mb/s. Nowadays, the 5th Generation (5G) of cellular wireless communication is taking place. Researchers now are focusing on utilizing these 5G technologies to improve the vehicular networks, which led to the invention of a new term, Vehicle-to-Everything (V2X) communication suite.

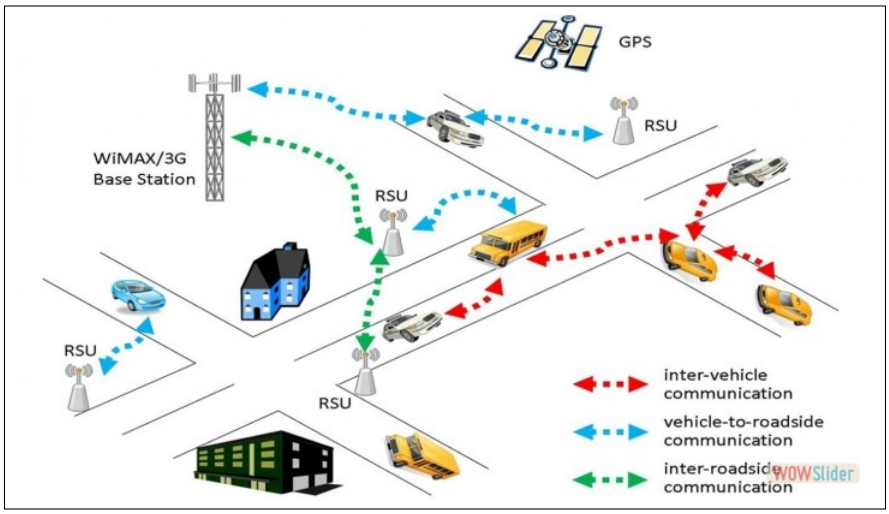


Figure 2.1: VANET [1]

2.1.2.2 Vehicle-to-Everything Communication (V2X)

After several years of VANET being in operation, it was realized that V2V communication only was not enough to enable a fast and effective deployment of most of the

application scenarios proposed for the vehicular networks. Therefore, it was realized that vehicles should not restrict their communications to be between each other only and some kind of extension was required. So, to allow the vehicles to communicate with the core network, with the pedestrians and may be bikers, a new communication suite was established, the V2X suite.

V2X is a set of technologies that allow a vehicle to communicate with the surrounding environment (see Figure 2.2) . There are several kinds of communication that V2X comprises like vehicle-to-vehicle (V2V), vehicles-to-infrastructure (V2I), vehicles-to-pedestrian (V2P) and vehicle-to-network (V2N). Use cases for V2X includes, but not limited to, platooning, collision alerts, road work warning and intersection movement assist. There are two types of technologies that V2X can rely one. The traditional WLAN-based (DSRC) and the LTE-based.

Figure 2.2: V2X [2]

To replace DSRC, the third generation partnership project (3GPP) developed the LTE-based communication technology called cellular vehicles-to-everything (C-V2X). Some of the benefits of replacing DSRC with C-V2X :

1. Cost-effectively, it allows the integration of the LTE network with the vehicular network.

2. It provides more comprehensive global positioning systems (GPS).

3. It improves safety by enabling drivers to have a more comprehending view to the street

4. It enhances the ability to avoid congestion and traffic jams.

5. It is more reliable in message passing between the vehicles.

2.1.3 Cooperative Autonomous Driving

In order for an AV to accomplish an on-the-road task, it needs to communicate with several entities including other vehicles, motorcycles, bicycles and even pedestrians. Also, IoT devices deployed over the street can provide the AV with potentially valuable information that assist the AV with it instantaneous maneuvering. **Cooperative Autonomous Driving** is *the concept of making automated vehicles interact directly or indirectly with each other and other entities such RSUs and IoT devices in order to accomplish certain on-the-road activities in a safe and effective way* [3]. This interaction is supposed to take place over the 5G-V2X networking protocols suite. With their promised high communication rate, these protocols are the platforms that Cooperative Autonomous Driving can be built over as discussed in several recent works [3]. The Society of Automotive Engineers (SAE) has suggested a message set dictionary for standardizing messages exchanged in communications like as emergency vehicle alerts, intersection collision warnings, and vehicle status information. Several CAD projects are in progress like the European Telecommunication Standard Institute (ETSI) provided the EN 302 637-2 standard which defined Cooperative Awareness Messages (CAMs). According to [3], the following are typical CAD use cases:

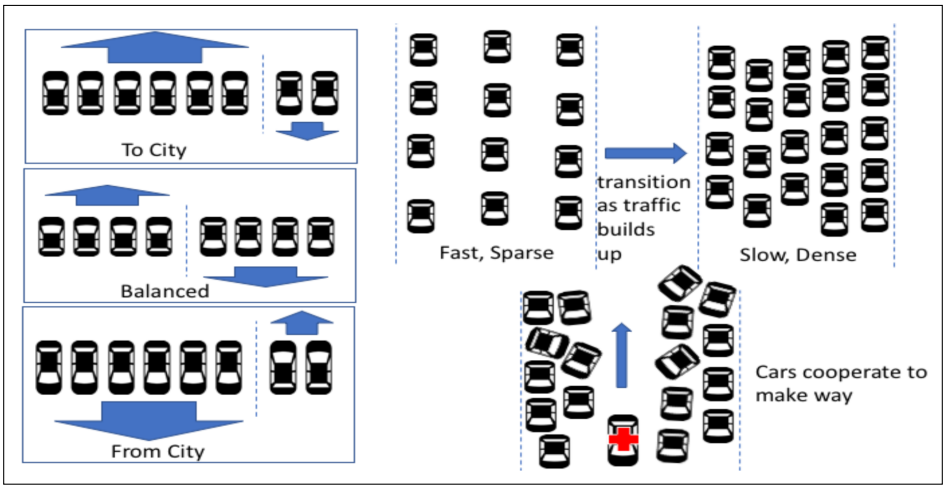


Figure 2.3: Dynamic traffic flow through swarms [3]

1. **Swarm behaviour for dynamic traffic flow**: The idea here is to avoid using physical signals (e.g., lights) in order to minimize the response time and increase the flexibility of the overall vehicles interaction (see Figure 2.3). A typical urban road consists of several lanes, half of them are for one direction. It is very common situation is when one direction on the road is more congested than the other. CAD mitigates such problem by making the vehicles cooperatively dedicate more lanes to the congested direction in order to have smooth traffic flow. Another scenarios where swarm behaviour is very beneficial is when we have an emergency vehicle requesting a path. CAD can easily establish a clear path by making the vehicle cooperate in creating such a path rather than the chaotic situation when each vehicle is making an independent decision.

2. **Platooning**: A platoon is a structure where several vehicles establishes in order to have a safer trip and reduce travel time and road usage. The idea is to have all the vehicles to travel very close to each other and the up front vehicle controls the direction and the speed. CAD here can be very beneficial to first, establish the platoon, and second maintain it through the vehicles trip.

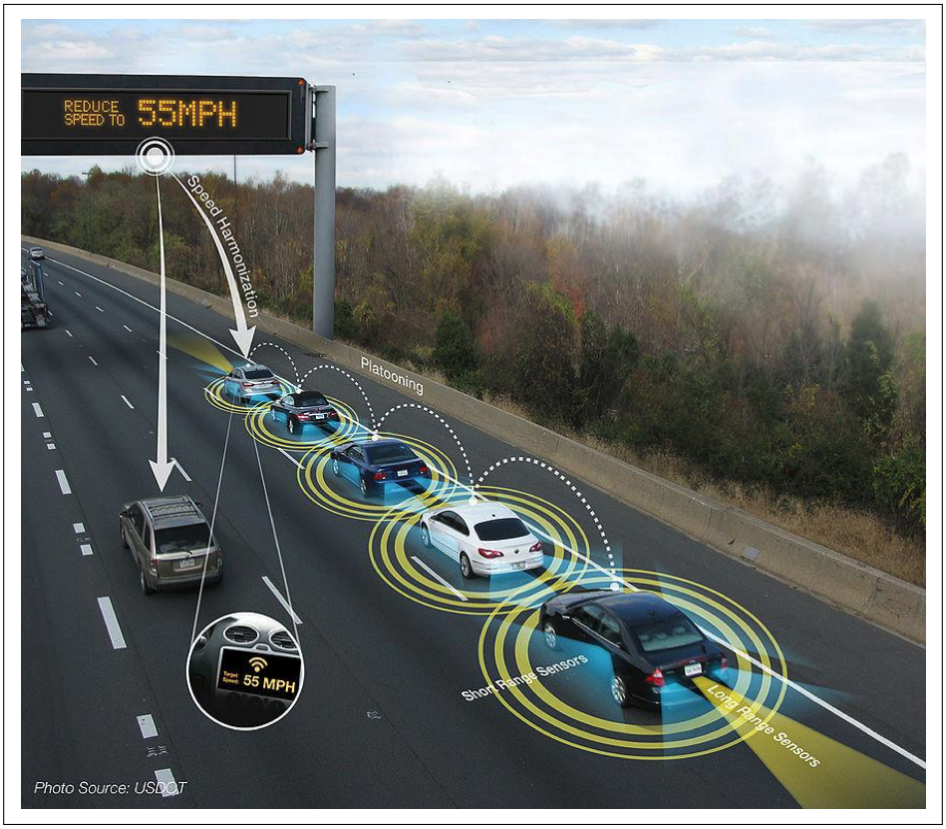


Figure 2.4: Vehicles in platoon form [4].

3. **Intersection**: Congested intersection is a critical problem that we all face in daily-basis. A promising solution is to make these intersections operate without traffic lights. CAD here plays the major role of controlling the flow of the intersected streets by signaling the vehicle when to stop and when to pass the intersection.

4. **Parking**: A vehicle can request information regrading the availability of the vehicles on a certain location.

5. **Routing**: Vehicles can cooperate to distribute them selves across several routes in order to avoid congested street hence minimizing the travel time.

2.2 Related Work

2.2.1 Vehicular Edge Computing

Previous literature explored several aspects of vehicular edge computing. For general system architecture and transmission strategies, [21] suggested a scheme to build a framework for a vehicular edge computing system consisting of multiple RSUs and several computation resources. This work concluded with an efficient transmission strategy that reduces the V2V and V2I transmission costs while maintaining a good transmission speed. The work in [17] proposed a communication protocol for vehicular multi-access edge computing by integrating licensed Sub-6 GHz band, IEEE 802.11p, and millimeter waves communications to distribute data in vehicular networks.

For resources management, the work in [22] proposed a technique that exploits the fact that multiple vehicles are accessing the same data stored at an RSU or in the cloud and suggested a scheduling scheme based on that. In [23], authors proposed a system model that consists of a set of RSUs each has its server with a certain computation capacity. If the computation overhead exceeds the server capacity, it can offload some tasks to a backup computation server. Each task has its upload data size, processing time, and deadline. A car can choose between uploading a task and running it locally based on the price of computing this task on the server. They assumed the speed of the communication between the servers and the backup server is infinite. The authors suggested a game theory approach to solve the problem. Another tasks-scheduling scheme is suggested in [24]. Their method tried to solve the problem of offloading tasks to multiple RSUs. Each type has a specific resources requirement. The service provider tries to increase his/her revenue while the vehicles try to reduce the power consumption and increase their computation speed. Another

similar work proposed in [25] considered a more general case where tasks with predefined processing time. The work proposed almost the same system model as the in [24] but without a backup computing server. A framework to exploit the idle time in the vehicles' computation resources is suggested in [26]. The work in [27] suggested resources management and allocation with power consumption reduction. The authors of [28] noted that MEC is a promising approach to allow resource-intensive services and presented a detailed futuristic vehicular scenario "the Electronic Horizon" and listed the challenges. The authors highlighted that Information-Centric Networking in combination with MEC could support such a futuristic scenario. In [29], the authors argued that recent advances in networking, caching, and computing have significant impacts on the developments of vehicular networks and proposed an integrated framework that can enable dynamic orchestration of the network resources based on deep reinforcement learning.

For mobile edge computing in general, several works addressed computational tasks scheduling. In [30], the authors explored the scenario of multi-cell mobile edge computing. Very similar work is proposed in [31]. An energy-efficient offloading scheme was described in [32]. Computation offloading in wireless cellular networks with MEC capabilities has also been studied in [33], and [34].

The 3rd Generation Partnership Project (3GPP) provided a communication standard for the vehicle-to-everything networks (V2X). These standards are supposed to support applications like vehicles platooning, and remote driving [35, 36]. The standards specified levels of driving automation that vary from "no automation" (level 0) to "full automation" (level 5). The channel's bandwidth can go up to 10 MHz in both ETSI and IEEE standards; hence the bit rate can reach up to 27 Mb/s and 54 Mb/s. Such bit rate can not support autonomous driving, which demands the latency to be no more than 100 ms. Hence, several works published recently proposed the

utilization of mmWave communication technologies for vehicular networks [37–39]. The only obstacle was the high dynamism of these kinds of networks, and the major contribution of these works was to overcome the high speed of the vehicles. In conclusion, [37] suggested that mmWave can support vehicles' speeds up to 140 km/h, and the provided bit-rate can reach 6.765 Gb/s. NOMA (Non-orthogonal multiple access) applicability for V2X was discussed in [40] and network slicing in [41].

2.2.2 Vehicular Fog Computing

Vehicular fog computing is taking considerable attention from academia, leading to extensive studies addressing major concerns. The work in [42] is a general survey discussing the motivations, the different architectures and issues of vehicular fog computing. In [43], they proposed a bidding-price-based mutual trust establishment between client vehicle and server vehicle and also payoff assignment based on transaction evaluation. The proposed method does not require of a trustworthy third-party. The work in [44] propsed a distributed information exchange scheme with low latency in vehicular fog computing. They considered the frequent changes in vehicle positions and used public transportation facilities such as buses and taxis as fog nodes. The fog nodes are responsible of adjusting the data sampling frequency according to the time-space correlation of the data to ensure that only nonredundant data are received. The interruption latency caused by accidents during an exchange is handled by evaluating and predicting connection states among the vehicles. In [45], authors used machine learning techniques to choose the best fog server deployed over a base station to be connected to a vehicle once it leaves a certain server range. The work considers the load and the location of the vehicles to accomplish an accurate prediction for the server. In [46], the authors suggested a machine learning algorithm that tries to utilize the mobility of the vehicle to minimize the delay of the tasks computation. The

work proposed architecture with three offloading modes, namely, vehicle-vehicle offloading, vehicle-RSU-vehicle offloading, and pedestrian-RSU-Vehicle offloading. The work discusses why mobility can be helpful to minimize the download time from a node to another. They considered two cases where, in the first, they combined an existing ML algorithm with coded computing to make it adaptable against the changes in the network topologies and workload. In the second, they investigated the idea of replicating the tasks to minimize the delay.

The authors in [47] suggested a three layers scheme to support traffic management. The three layers are the cloud, the cloudlet, and the fog layer. The fog consists of parked and moving vehicles having a certain computation capability. With this architecture, the work proposed an algorithm that balances the load over the layers resources by distributing messages passed by vehicles over the computation resources to process them. The work in [48] suggested a design principle for fog-enabled vehicular software-defined networking. The authors evaluated the design with the use case of a traffic management system for fast traffic rescue using real traffic accident data. In [49], authors tried to utilize the idle computation resources of the parked electrical vehicles. The problem was formulated as a Markov decision process and solved through dynamic programming. The work in [50] suggested an offloading scheme of tasks generated by the users' equipment to the vehicles based on contract theory and matching theory. The aim was to minimize the computation delay. A task has its required number of computational cycles, the upload data size, and the deadline. The download data size is assumed to be negligible, meaning, the computation's result can be downloaded instantaneously. They can be scheduled in a non-preemptive manner only. A vehicle offers its computational resources it is willing to share and assumed to have a fixed location. A vehicle can only be assigned one task. The work assumes a dedicated bandwidth assigned to each user's equipment. First, they designed a

contract relating the required performance levels to the payments issued to vehicles offering this performance. Then a two-sided matching game approach is performed to assign the tasks to the vehicles. In [51], authors proposed a method to take advantage of the possibility of dividing the tasks offloaded from vehicles into several subtasks (there is no constraint on how they can divide these tasks). The work first studied the status of the channels through the hidden Markov model and proposed a scheme to leverage the full parallelism of computational tasks. The authors in [11] provided a mathematical model that studies the vehicular fog computing capabilities quantitatively. The work used realistic data acquired from tens of thousands of taxis. For data distribution application, [52] suggested a joint optimization of access mode selection and spectrum allocation while considering the randomness of the vehicular network, the edge cache, and the content download delay. VeFC was applied for various applications like mobile crowdsensing [53], caching [54], traffic management [55]. The authors in [56] suggested an intriguing role of fog computing in the context of Information-Centric Networking (ICN). The work indicated an integrated fog-computing and ICN architecture with an on-demand caching function virtualization scheme and a communication scheme between the fog nodes and future internet nodes. They also designed an intelligent control to manage the operations between these nodes and a cognitive resource allocation.

2.2.3 Age of Information in Vehicular Networks

In [57], the authors studied the radio resource management in a V2V network. The work considers the Manhattan grid V2V network. They studied a dynamic traffic model and assumed the channel quality changes based on the geographical location. They modeled the problem as a single-agent MDP deployed over an RSU to allocate the radio resources, aiming to minimize AoI. In [58], rather than optimizing the

average AoI, the authors proposed to control the tail AoI distribution in V2V communication networks. In this work, the authors tried to minimize the probability of AoI going beyond a particular value. The work used the extreme value theorem and Lyapunov stochastic optimization technique to minimize the objective mentioned. The authors of [59] developed an approach that is based on Gaussian process regression to learn V2V system dynamics to estimate the AoI and accordingly allocate the transmission power while minimizing the tail AoI. In [60], the authors proposed to analyze and improve the tail of the probabilistic AoI in vehicular networks through the extreme value theory. They derived a relationship between the probabilistic AoI and the data queue size in each vehicle then used the result as a constraint in the optimization problem. The cars are clustered into groups to reduce interference. Their problem is formulated as a probabilistic scheduling problem and solved through a policy to assign each job to a certain virtual machine. The authors in [61] showed that in vehicular networks, reducing the system age cannot be achieved by maximizing the throughput in a practical 802.11 system. The work in [62] discussed the beacon broadcasting scheduling problem to minimize AoI. They solved the problem through a greedy heuristic. Optimizing the driving route to maintain the confidence of AoI is discussed in [63].

AoI in communication systems was considered earlier in several works. In [64] they studied the improvement of AoI while respecting the hard deadline of the data transmission. Authors of [65] suggested a scheduling policy for data arriving at a base station to transmit it to different destinations. Each stream has a single-item queue, and the aim was to minimize AoI. The authors of [66] proposed a similar work in which they considered an M/M/1 for each stream and proved that a queuing delay might not lead to an increase in the information age.

2.2.4 Cooperative Autonomous Driving

Several works in the literature discussed the support of CAD through edge computing. The work in [67] considers a scenario where several sensors are updating a server with data and requesting a computation at a specific rate. Both computation time and data freshness are optimized jointly. They solve the problem through two stages. The first stage is assigning a job to a virtual machine, which they solved through a probabilistic scheduling policy. The second is to minimize the completion time and AoI through the convex optimization technique. In [68], they studied the performance of inter-vehicle communication. The work explores the performance through a simulation of the network and convoys of automated vehicles. Authors of [69] utilized connected vehicles and infrastructure-assisted management to increase safety around highway work zones. The work used a manual driving simulation to test the proposed solution. The work in [70] offered an alternative to an earlier ETSI standard perception messaging to improve the vehicle's perception capabilities. The approach reduced the number of messages sent hence reducing the resources usage. In [71], they proposed a transition of control (from the vehicle to the driver) method to increase safety by informing the car (via the infrastructure) about the place it can safely stop.

Chapter 3

Workload Scheduling in Vehicular Networks with Edge Cloud Capabilities[1]

[1]This chapter has been published in IEEE Transactions on Vehicular Technology [72].

We consider a vehicular network with edge cloud capabilities; vehicles offload their workloads to cloudlets co-located with roadside units deployed along the road and providing computational capabilities for applications running on driving-by cars. We assume two types of applications, 1) critical and highly time-sensitive (e.g., control and safety), and 2) delay insensitive with higher bandwidth requirements. We consider a single RSU and find a solution for scheduling the workloads task processing, taking into account vehicles speeds, residence times within the coverage, and the delay requirements of the services. We mathematically model this problem to realize an efficient scheduling policy and provide proof of its NP-hardness. We subsequently propose a polynomial-time and efficient solutions based on the Lagrangian decomposition method and a greedy scheduling method.

3.1 Motivation

To cope with the explosive computation demands of vehicular nodes, cloud-based vehicular networking has emerged as a very promising concept to improve the safety, comfort as well as the experience of passengers. By integrating communication and computing technologies, cloud-enabled RSUs allow vehicles to offload their tasks that require high computational capabilities to the remote computation cloud, thus undermining the shortcomings of limited processing power and memory capacities of a vehicle's OBU. This scenario is widely known as Mobile Cloud Computing (MCC), which greatly improves resource utilization and computation performance and provides several advantages including, but not limited to, *1)* extending the mobile's battery lifetime by offloading energy consuming computations to the cloud, *2)* enabling sophisticated memory-exploiting applications to the mobile users, and *3)* providing higher data storage capabilities to the users. However, considering the capacity limitation and delay fluctuation of the transmission on the backhaul and backbone

networks, placement of the cloud servers far away from the mobile vehicles may cause serious degradation of the offloading efficiency. As such, Vehicular Edge Computing (VEC) is proposed as a promising motion that pushes the cloud services to the edge of the radio access network, namely the RSU, and provides cloud-based computation offloading within the RSU's communication range.

3.2 Contributions

We can summarize the contribution of this work as:

- We formulate task offloading as preemptive scheduling to maximize the weighted sum of admitted tasks (throughput). We prove the problem is NP-hard through a reduction from the multiple knapsack problem.

- To meet the latency requirement of vehicular workload, maximize the number of admitted tasks, and improve the quality of service, we propose two efficient methods to solve the formulated scheduling problem. The methods are based on the Lagrangian relaxation technique, which finds near-optimal solutions much faster than the CPLEX (see Appendix B.2) branch-and-cut method. We also offer a greedy heuristic to solve the problem.

- We formally prove that both Lagrangian relaxation methods provide a better upper bound than the standard linear relaxation. We prove also that one of the methods can reach the optimal solution.

- The suggested methods are evaluated empirically by generating random instances, using traces generated by SUMO (see Appendix B.1), and assessing the performance in terms of the execution time and rejection rate.

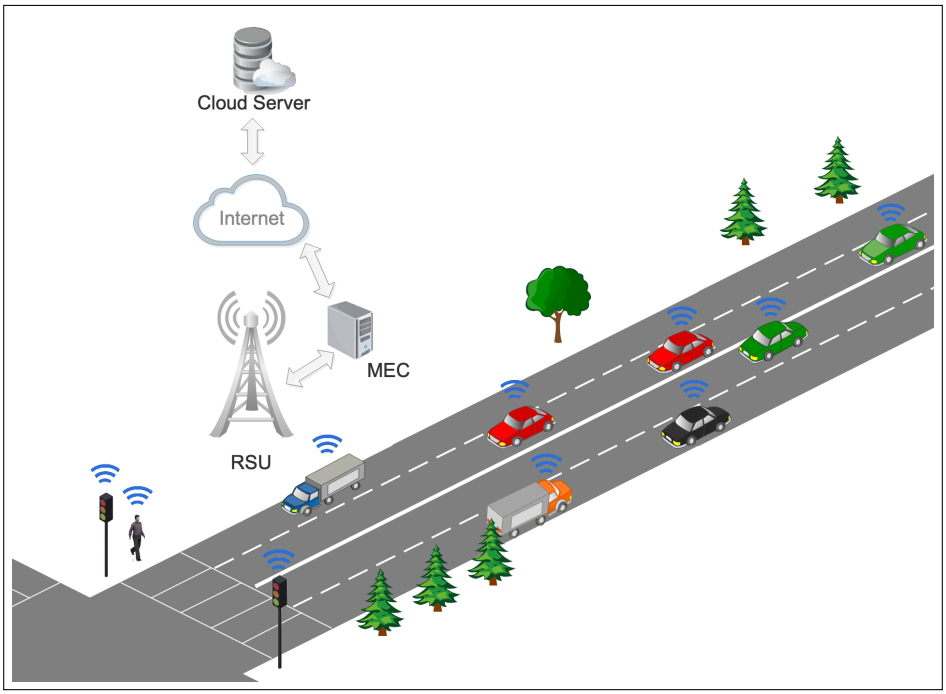


Figure 3.1: Example of a vehicular network task offloading over a single RSU with k computation resources.

3.3 System Model

As shown in Figure 3.1, the proposed system model consists of vehicles navigating along a roadway segment. These vehicles may request the RSU with VEC capabilities to process their tasks which require high computational power. The RSU is assumed to possess multiple computation resources with high processing power. We suppose that each vehicle is equipped with an OBU to communicate with the RSU according to the Wireless Access for Vehicular Environment (WAVE) protocol suite, particularly the Dedicated Short Range Communication (DSRC) standard. The WAVE communication spectrum allows nodes to periodically broadcast beacon messages over the Control Channel (CCH) announcing their offered services (in case of an RSU) or information about their speed, location, processing requests, etc. (in case of a vehicle). In this work, a vehicle selected to upload its request or download the results of its requested task coordinates with the RSU and switches to a Service Channel (SCH)

to establish a communication link.

Given that vehicles have limited residence time within the RSU communication range, it becomes crucial to manage the RSU's VEC resources efficiently. Therefore, it respects the deadlines of delay intolerant tasks, and 2) completes the maximum number of delay-tolerant tasks. It is worthwhile mentioning that one might argue that it could be more efficient to transfer delay-tolerant tasks to the mobile cloud rather than the VEC server. However, one must note that these tasks consume a large bandwidth and experience longer delays. Hence, the vehicular network should exploit its edge capabilities rather than depending on central cloud servers to process computational tasks. The VEC should efficiently schedule the processing of these tasks to respect their delay tolerance (later represented as task deadline) and minimize the number of rejected tasks.

At the beginning of each scheduling epoch, the RSU with VEC capabilities collects all information of the set of in-range vehicles and their associated computational tasks which they wish to offload. Recall that the RSU keeps track of the vehicles' arrival times and speeds, and hence, their respective remaining residence times. Now, the RSU can perform preliminary filtering and directly deny the requests that VEC cannot process due to two reasons. *1)* The task requires more resources than the RSU has, or *2)* the time necessary to compute the task is greater than the remaining residence time of the requesting vehicle. After completing this pre-processing step, the RSU now has a fully deterministic representation of the system and can schedule the processing of computational tasks on its available servers. But, it should consider that VEC should communicate the result of an admitted task to the requesting vehicle before that latter leaves the communication range of the RSU. The scheduler also counts the time required to upload the necessary data for the task computation and download the result of those computations. This work proposes a scheme composed of four

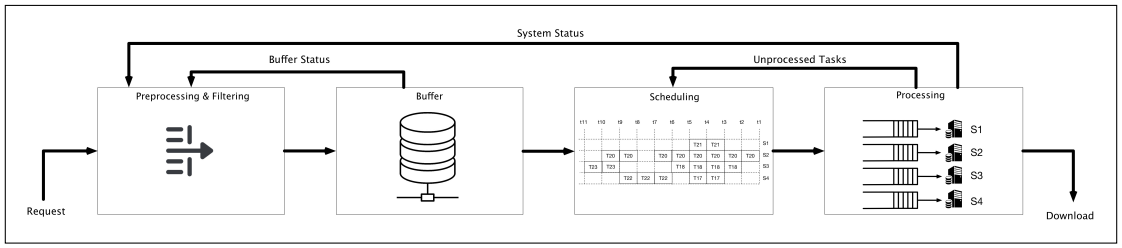


Figure 3.2: The system scheme

stages that fully describe the dynamics of the adopted system.

As depicted in Figure 3.2, for each scheduling epoch, all requests pass through a pre-processing and filtering stage where a subset of requests VEC has naturally eliminated due to the previously mentioned reasons. Next, the RSU runs a scheduling algorithm and decides which requests will be processed. Then VEC will advise vehicles initiating these requests to upload their requests immediately. Then, the requests will either occupy a computational server or wait in a buffer until it is their time to be processed[2]. Once completed, the system will download the result of a requested task to the initiating vehicle. Now, it is true that in the forthcoming scheduling epoch, some previously admitted requests are still in the buffer waiting to be processed. To maintain high system reliability, the RSU has two choices. 1) It locks the resources until they process previously admitted requests and runs a scheduling algorithm to schedule the newly arriving requests with the remaining resources. 2) It re-schedules these previously admitted requests along with the newly arriving requests. Still, this time, it forces the scheduler to accommodate buffering tasks and deny any reneging of previously accepted tasks. The following two subsections present in detail the pre-processing and buffering stages.

[2]Each server can be assumed to be a virtual machine with fixed processing capacity running on the edge cloud. A request occupies a VM until it finishes processing without sharing the resources. The number of VMs is known ahead of time.

3.3.1 Tasks Preprocessing and Filtering

Pre-processing and filtering of the tasks are the first steps performed at the beginning of each scheduling epoch. At this stage, the RSU has collected all information required to decide whether VEC can process the received request given its requirements and the departure time of the initiating vehicle. Let T_i^{total} be the total time required to upload a task i (including making a request), process it, and then download the result of the computation to the requesting vehicle. T_i^{total} is given by:

$$T_i^{total} = max(T_i^b + T_i^s, T_i^u) + T_i^p + T_i^d \tag{3.1}$$

Where T_i^b is the time the request spends in the buffer waiting to be processed, T_i^s is the time of scheduling the tasks, T_i^u is the time required to upload the task, T_i^p is the time required to process the data and T_i^d is the required download time. We may neglect the preprocessing time as we can obtain its results after solving simple equations. We count, therefore, only the maximum of $T_i^b + T_i^s$ and T_i^u. That is because the vehicle will start offloading the required task once its request passes the preprocessing and filtering stage; hence, these two values overlap, and we should consider only their maximum. We will discuss the time a request spends in the buffer waiting to enter the processing stage in the next subsection. The upload and download times depend on the wireless channel between the vehicle and the roadside unit. Let $r_i(t)$ be the rate achieved by a vehicle i at time t, where $r_i(t) = Blog_2(1 + SNR_i(t))$. Here, B is the channel's bandwidth, and $SNR_i(t)$ is the signal to noise ratio at the receiver (e.g., RSU) at time t. Given that cars are moving at high speeds, we assume the achieved rate varies vs. time within the coverage of the RSU. Namely, $SNR_i(t) = \frac{p_i \times g_i(t)}{N}$, where $g_i(t)$ is the channel gain to the RSU, which depends on the distance between

vehicle i and the RSU (we assume a path loss channel, and hence $g_i(t) = (d_i(t))^{-\sigma}$, σ being the path loss exponent), p_i is the transmit power of the onboard unit module and N is the noise term. It is clear that the upload and download times are dependent on the instantaneous location of the vehicles within the coverage of the RSU; the closer the vehicle is to the RSU, the higher the rate and the farther it is from the RSU, the lower is the transmit rate (which affects T_i^{total}). We adopt a widely used model for describing a task, where a task i has a data size u_i (e.g., in bytes) and a computation requirement Q_i (e.g., number of CPU cycles). In addition, each task specifies a deadline α_i for its computation to complete (e.g., self-driving cars require very tight response time), as well as a priority w_i (e.g., to distinguish tasks of different applications). Therefore, the upload time $T_i^u = \frac{u_i}{r_i}$ where:

$$u_i = \int_{t_0}^{T_i^u + t_0} r_i(t)dt \tag{3.2}$$

Here, t_0 is the time a vehicle starts offloading the task. $r_i(t)$ contains the term $d_i(t)$, where $d_i(t) = \left[(a_x - t \times v_i)^2 + a_y^2\right]^{0.5}$. Here, v_i is the vehicle's speed, t is the time that has elapsed from the start of the upload time, and (a_x, a_y) are the 2D coordinates of the base stations, (0,0) being the entry point to the coverage of the RSU.

Solving the above complex equation can be avoided by dividing the entire RSU coverage into multiple zones, as illustrated in Figure 3.3; and we assume that the transmission rate of each zone is constant and calculated using $r_i(t)$ where the distance used is the distance from the RSU to the centre of the zone. Let $s \in S = \{1, 2, \ldots, \mathcal{K}\}$ be the index of a subrange. Each subrange has its own data rate r_s, a starting point f_s and an end point e_s ($e_s - f_s$ is the length of the subrange covered by a rate r_s). Hence, the maximum amount of data that will be uploaded in subrange s, say u_{is}, is given as $u_{is} = \frac{(e_s - f_s) * r_s}{v_i}$.

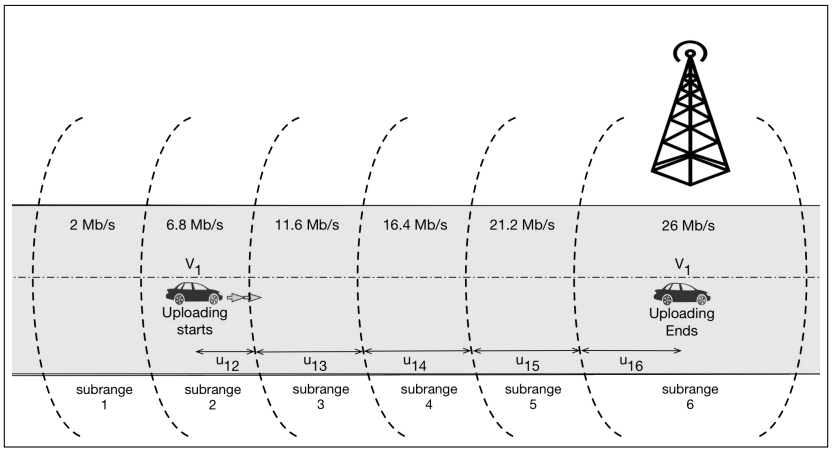


Figure 3.3: Vehicle 1 uploading starting from subrange 2 to subrange 6

Let $l_i \in [-R, R]$ to be the initial location of vehicle i, then T_i^u can be calculated as shown in Procedure 1.

Algorithm 1 Computation of T_i^u

$s \leftarrow s_i$
while $u_i > 0$ **do**
    $u_{is} \leftarrow \frac{(e_s - l_i) * r_s}{v_i}$
    if $u_i - u_{is} > 0$ **then**
        $T_i^u \leftarrow T_i^u + \frac{e_s - f_s}{v_i}$
    else
        $T_i^u \leftarrow T_i^u + \frac{u_i}{r_s}$
    $u_i \leftarrow u_i - u_{is}$
    $s \leftarrow s + 1$
    $l_i \leftarrow f_s$

Now, to evaluate the time required to download the result of processing a requested task, let d_i be the data size to be downloaded after processing task i. The maximum amount of data that the system can download in subrange s is equal to the maximum amount of data that a vehicle can upload in that same range since the adopted data rate is the same. As such, We can obtain T_i^d following the same method used to compute T_i^u.

The departure time of vehicle i, say β_i, can be easily evaluated since its velocity v_i, location l_i, and the antenna range R are known. But since the generating vehicle

must receive the outcome of computation before it departs the RSU range, we should take the task download time into account. To ensure that the RSU transmits the result of the computed task to the generating vehicle, the latter's departure time is modified. It accounts for the download time of the task, assuming that the vehicle completes its download just before it leaves the RSU communication range. As such, we can calculate β_i as follows:

$$\beta_i = \frac{R - l_i}{v_i} - \tilde{T_i^d} \tag{3.3}$$

where $\tilde{T_i^d}$ is the (maximum) time required by vehicle i to download the result of its request as it approaches the end of the RSU communication range. $\tilde{T_i^d}$ can be calculated similarly to the upload time, but starting from the last subrange and iterating over the preceding subranges. In other words, $\tilde{T_i^d}$ represents the worst case where the vehicle gets its computation result towards the end of the coverage range of the RSU. The deadline of the task, α_i, must go through the same calculations to make it embrace the download time.

Hence, it is now clear how the RSU filters out the requests that VEC cannot process before the departure of the initiating vehicle. A vehicle whose request passes the filtering stage for possible processing uploads its relevant data right after the pre-processing step. During this time, the VEC server is solving the scheduling problem, and received tasks will wait in a buffer until the scheduling policy is realized. Let γ_i be the release time of task i. γ_i is the time at which the RSU receives task i completely. If the RSU receives a task during the buffer period of a scheduling epoch, we set its release time to 0. It means that VEC may schedule this task to start running right after the buffering period ends. Otherwise, its release time is the remaining time required for the RSU to receive it completely. Once the RSU has all the required information to schedule the processing of accepted tasks, it engages in realizing a

scheduling policy. The following subsection discusses how frequently the scheduling policy is re-visited and modified.

3.3.2 Tasks Buffering and Scheduling

Clearly, it is impractical to re-visit and modify the scheduling policy whenever the RSU receives a request. Therefore, the tasks that complete the pre-processing and filtering step are buffered for some time before the RSU engages in realizing a new schedule. We referred previously to this time as buffering period. Efficient tasks scheduling is the ultimate objective of this work. We establish the optimal scheduling policy by solving an ILP model. However, the latter solution does not scale well. It may be infeasible, especially in a delay-intolerant environment, since the time required to solve the model increases tremendously as the number of requests increases. For this purpose, this work proposes scheduling approaches based on Lagrangian relaxation techniques to solve the underlying ILP problem efficiently. Finally, it is important to mention that each scheduling epoch must account for previously admitted tasks. In other words, if scheduling epoch s_t admitted task i, then scheduling epoch s_{t+1} should be forced not to reject task i. We discuss the scheduling approaches in detail in the next section.

3.4 VEC Workload Scheduling

Scheduling the offloaded tasks by vehicles is the major problem the VEC system must tackle. Our system requires scheduling two sets of tasks. Tasks that are already in the server's queues (pre-admitted tasks) and the newly arrived ones. In the following subsections, we provide the problem definition and mathematically model it as an ILP.

3.4.1 Problem Definition

Given a set of N tasks $\mathcal{I} = \{1, 2, ...i..., N\}$ each has a processing time[3] p_i (sec), release time γ_i, deadline α_i, departure time β_i and a weight w_i. In addition, a set of N' pre-admitted tasks $\mathcal{I}' = \{1 + N, 2 + N, ...i + N..., N + N'\}$. We want to schedule these two sets of tasks over a single RSU with edge computing capabilities and having M computation machines. We want to serve most new tasks and guarantee the rescheduling of the pre-admitted ones after their release times and before the minimum of the deadline and departure time. The calculations of $\{\alpha_i\}$, $\{\beta_i\}$ and $\{\gamma_i\}$ were explained in section 3.3. The obligation of rescheduling the pre-admitted tasks embodies the problem as a special case of a general scheduling problem described as $P|pmtn, r_i|\sum w_iU_i$. In words, it is *"scheduling jobs with release time, deadline and processing time on identical parallel machines to minimize the weighted number of rejected jobs"* [73].

Proposition 3.1. *The problem $P|pmtn, r_i|\sum w_iU_i$ is a strong NP-Hard problem.*

Proof. Consider an instance of the multiple knapsack problem (MKP) where there are M bins $B = \{1...b...M\}$ each with its own capacity, say c_b, and N articles $A = \{1...a...N\}$ each has a gain g_a and size s_a. The objective of the MKP is to maximize the profit of packed articles in the bins in such a way the sum of packed articles sizes in each bin does not exceed the bin capacity. To reduce an MKP instance into our scheduling problem, we do the following: For each article a, create a task i with weight equals to g_a, processing time equals to s_a, deadline equals to $\max\limits_{1\leq j\leq M}\{c_j\}$ and release time equals to 0. For each bin b, create a machine. If the bin does not have the maximum capacity in the MKP instance, create a task t_b with a processing time equals to $\max\limits_{1\leq j\leq M}\{c_j\} - c_b$, its deadline is $\max\limits_{1\leq j\leq M}\{c_j\}$, its release time is c_b and its weight is $\sum_{a\in A} g_a + 1$.

[3] p_i maybe derived from Q_i and the capacity of each VM.

The purpose of the tasks $\{t_b\}$ is to reduce the timeline of the machines to make it equal to the corresponding bin size in the MKP instance. By making their weights larger than the weights of all the other tasks combined, and there is enough time to schedule each in one machine, the optimum solution *must* contain all the tasks in $\{t_b\}$.

The reduction must address two more concerns. First, in MKP, the articles cannot be partitioned while the tasks in the scheduling problem can be scheduled in a non-preemptive way. Second, an article can be packed only in one machine while the task can be scheduled in multiple machines. To address the first, we make the lifespan of all the tasks the same (i.e., the deadlines of the tasks are equal and their release times). By doing so, it will be unnecessary to schedule a task preemptively. Even if that happens, it will be easy to re-organize the tasks, so they become one piece. We solved the second concern by solving the first one. If a task is scheduled in more than one machine, we can re-organize the tasks schedule, so we get each task scheduled in one machine. This completes the reduction. Because the reduction runs in polynomial time and MKP is a *strong NP-Hard* problem, then $P|pmtn, r_i|\sum w_i U_i$ is also a *strong NP-Hard* problem. □

The proof does not include any pre-admitted tasks because, in MKP, there is no requirement to enforce the packing of specific articles. Owing to its complexity, we next look for low complexity methods for solving the workload scheduling problem. However, we first start by mathematically modeling the problem as an integer linear program (ILP), which we will use to derive two efficient methods following the Lagrangian relaxation framework.

3.4.2 Mathematical Model

Let x_i be a binary variable that decides whether task i is admitted or not. Also, let y_{it} be a binary variable that decides whether to schedule task i in time slot t or not. Then the model of the general $P|pmtn, r_i|\sum U_i$ problem is the following:

$$maximize \sum_{i \in \mathcal{I}} w_i x_i$$

$$s.t.$$

$$\begin{aligned} y_{it} = 0 \quad \forall i, t \quad &: \quad t \geq min(\alpha_i, \beta_i) \\ &\vee \quad t \leq \gamma_i \end{aligned} \tag{C1}$$

$$\sum_{t \leq T} y_{it} = p_i x_i \quad \forall i \in \mathcal{I} \tag{C2}$$

$$\sum_{i \in \mathcal{I}} y_{it} \leq M \quad \forall t \leq T \tag{C3}$$

$$x_i \in \{0, 1\} \quad \forall i \in \mathcal{I}$$

$$y_{it} \in \{0, 1\} \quad \forall i \in \mathcal{I} \;\; \forall t \leq T$$

The objective of the model is to maximize a weighted sum of the number of admitted tasks. Constraint C1 makes sure that no task is scheduled after its deadline or before its release time. Notice that the deadline of a task is whatever comes first, the departure time β_i or the deadline α_i. C2 enforces the scheduler to either schedule a task and complete it or does not schedule it at all. C3 prevents scheduling a number of tasks more than the number of computation utilities on one unit of time. The mathematical model described above does not model the requirement of re-scheduling pre-admitted tasks. We next propose two approaches. The first is to add the following constraint:

$$\sum_{t \leq T} y_{it} = p_i \quad \forall i \in \mathcal{I}' \tag{3.4}$$

This constraint is very similar to constraint $C2$. It forces the algorithm to re-schedule all pre-admitted tasks, but not necessarily on the same time units. In addition, C3 must now include $\{y_{it}\}$ of the pre-admitted tasks in its summation. We call this technique *re-scheduling* the pre-admitted tasks. The other solution is to *fix* the pre-admitted tasks in the same schedule before the arrival of the new task and schedules the new tasks by avoiding its overlap with the pre-admitted ones. We can do this by making the number of machines available dependent on the time unit (i.e., M_t instead of M). This dependency allows the scheduler to assign machines to the new tasks while reserving others to the pre-admitted tasks in each time unit.

The next section will describe two approaches to solve the problem efficiently. Both are based on Lagrangian relaxation. In the subsequent sections, we refer to constraint C2 as the *integrality constraint* and C3 as the *capacity constraint*.

3.5 Lagrangian Relaxations

Lagrangian relaxation is a well-known technique to relax a model into a simpler one by moving one or multiple constraints to the objective function and multiply each with its corresponding Lagrangian dual variable ($\boldsymbol{\lambda}$). The dual variable essentially represents a penalty for violating a constraint. The relaxed model usually is easier to solve (i.e., polynomial-time solvable). So the technique solves the relaxed model multiple times with different values of $\boldsymbol{\lambda}$'s until it reaches a tighter upper bound to the original problem. Every time the relaxed model is solved, the method must explore the possibility of constructing a lower bound solution. The technique keeps iterating until it reaches a specific convergence criterion. In most LR implementations, the

sub-gradient descent is the method that updates the dual variables.

Looking at our scheduling problem (ILP), although it has three constraints (C1, C2, and C3), it is a complex problem to solve. This characteristic lets us figure out which constraint(s) makes the ILP program hard to solve. The Lagrangian relaxation technique allows us to exploit the characteristic of each constraint and simplify the problem by relaxing a proper one without going very far from the original problem. The following two subsections will describe the Lagrangian relaxation of our ILP problem by relaxing the two constraints C2 and C3, respectively.

3.5.1 The Integrality Constraint Lagrangian Relaxation (ICL)

In this subsection, we start by relaxing constraint C2 (integrality constraint). The following model represents the Lagrangian relaxation of the integrality constraint while rescheduling the pre-admitted tasks:

$$
\begin{aligned}
L_{c2}(\boldsymbol{\lambda}) = \Big(& maximize \sum_{i \in \mathcal{I}} (w_i - \lambda_i p_i) x_i \\
& + \sum_{t \leq T} \sum_{i \in \mathcal{I} \cup \mathcal{I}'} \lambda_i y_{it} \\
& - \sum_{i \in \mathcal{I}'} \lambda_i p_i \\
s.t. & \\
y_{it} = 0 \quad \forall i, t \quad & : \quad t \geq min(\hat{\alpha}_i, \beta_i) \\
& \vee \quad t \leq \gamma_i \\
\sum_{i \in \mathcal{I} \cup \mathcal{I}'} y_{it} \leq M \quad & \forall t \leq T \\
x_i \in \{0,1\} \quad \forall i \quad y_{it} \in \{0,1\} \quad & \forall i, t \Big)
\end{aligned}
$$

Algorithm 2 Solution for the relaxed ICL problem

1: **procedure** ICL-UPPER-BOUND($\boldsymbol{\lambda}, \boldsymbol{w}, \boldsymbol{p}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, M$)
2: $x_i \leftarrow 0 \quad \forall i \in \mathcal{I}$
3: $y_{it} \leftarrow 0 \ \forall i \in \mathcal{I} \cup \mathcal{I}' \quad \forall t \leq T$
4: **for** $i \in \mathcal{I}$ **do**
5: **if** $\lambda_i p_i < w_i$ **then**
6: $x_i \leftarrow 1$
7: $\mathcal{I}_s = sortIndexDesc(\boldsymbol{\lambda})$
8: **for** $t = 1$ to T **do**
9: $M_t = 0$
10: **for** $i \in \mathcal{I}_s$ **do**
11: **if** $t \geq \gamma_i$ & $t < \min(\alpha_i, \beta_i)$ & $M_t < M$ & $\lambda_i \geq 0$ **then**
12: $y_{it} = 1$
13: $M_t = M_t + 1$
return $\{\boldsymbol{x}, \boldsymbol{Y}\}$

We propose algorithm 2 to solve the Lagrangian relaxation of the problem $L_{c2}(\boldsymbol{\lambda})$. In lines 4-66, we set x_i to 1 if it has a positive term (i.e., $w_i - \lambda_i p_i > 0$), so as to increase the objective function. For the variables set $\{y_{it}\}$, we sort all the tasks $(\mathcal{I} \cup \mathcal{I}')$ in descending order based on their dual variables $\{\lambda_i\}$ and store them in $\mathcal{I}_s$ (line 7). Then, for each time unit $t \leq T$ (line 8), we check for each task i in $\mathcal{I}_s$ (line 10) whether it satisfies its deadline (γ_i), departure time ($\min(\alpha_i, \beta_i)$), and whether it has a positive dual value (i.e., $\lambda_i \geq 0$), and also whether there is a machine available for processing (lines 11). If yes, we assign task i to that machine and set $y_{it} = 1$ (line 12).

Proposition 3.2. *Procedure 2 returns the optimal solution of the relaxed model ICL(λ).*

Proof. Relaxing the integrality constraint (C2) allows us to optimize the two sets of variables, $\{x_i, \forall i \in \mathcal{I}\}$ and $\{y_{it}, \forall i \in \mathcal{I} \cup \mathcal{I}' \& t \leq T\}$ independently. For any x_i, as it can be seen from the relaxed optimization model $L_{c2}(\boldsymbol{\lambda})$, since this variable is independent from others (i.e., no constraint is restricting the relationship), it can be optimized independently as explained above. Regarding $\{y_{it}\}$, notice that each

variable in this set appears only in one constraint of the sets $C3$. Hence, each group of $\{y_{it}\}$ variables that appear together in one constraint can be optimized independently. In essence, there is a knapsack problem in each time unit where the items are the tasks, and the capacity is the number of machines available. But since all the items (tasks) have the same size, algorithm 2 should choose the ones (i.e., set $y_{it} = 1$) that maximize the term $\sum_{i \in \mathcal{I} \cup \mathcal{I}'} \lambda_t y_{it}$ (lines 7-13). □

Proposition 3.3. *The time complexity of the ICL algorithm is* $O(T\hat{N} + \hat{N}log(\hat{N}))$, *where* $\hat{N} = N + N'$.

Proof. Initially, it takes $O(N)$ to check N tasks (x_i, $i = 1, 2, ..., N$) whether they have a positive term (i.e., $w_i - \lambda_i p_i > 0$), and assign a value for each task. Sorting all tasks in the set $\mathcal{I} \cup \mathcal{I}'$, based on their dual values, takes $O(\hat{N}Log(\hat{N}))$. Now, for each time unit $t \leq T$ and for each task in the sorted list $\mathcal{I}_s = \mathcal{I} \cup \mathcal{I}'$, we need to check whether to set the value of y_{it} to one or not, which takes $O(T\hat{N})$. Therefore, in total it takes $O(N + T\hat{N} + \hat{N}log(\hat{N}))$ to run the algorithm. Note that $N \subset \hat{N}$. Hence, the time complexity of the ICL algorithm is $O(T\hat{N} + \hat{N}log(\hat{N}))$ □

3.5.1.1 Obtaining a feasible solution for the problem

So far, the result obtained from Procedure 2 is an upper bound for the problem. To get a feasible solution (i.e., a lower bound solution), we have to make sure that all the pre-admitted tasks have been scheduled. This requirement can not be guaranteed by algorithm 2. If this condition is not satisfied yet, we give higher initial dual values to the pre-admitted tasks to let the scheduler schedule them first. Since these pre-admitted tasks were already scheduled in the previous epoch, it is guaranteed that they will be scheduled in the current epoch. Another solution is to fix the scheduling of the pre-admitted tasks from the last epoch, and hence, let the scheduler solves for the newly arrived tasks on available machines and time units.

3.5.1.2 Updating dual variables using sub-gradient descent

let vector $S =< s_1, s_2, s_3, ..., s_{\hat{N}} >$, where $s_i = \sum_{t \leq T} y_{it} - p_i x_i$, and $x_i = 1(\forall i \in \mathcal{I}')$. Then the dual variables $\{\boldsymbol{\lambda}\}$ are updated using the following equation:

$$\boldsymbol{\lambda}^{\tau+1} = \boldsymbol{\lambda}^{\tau} + \frac{\epsilon * [FS(\boldsymbol{\lambda}^{\tau}) - L_{c2}(\boldsymbol{\lambda}^{\tau})]}{\|S\|^2} S \tag{3.5}$$

where, τ denotes the current iteration, $FS(\boldsymbol{\lambda^{\tau}})$ is the obtained feasible solution, and ϵ is a constant in the range of $[0, 2]$. Notice that the nominator of the second term is always negative because the feasible solution is the lower bound of the objective function. The Lagrangian is the relaxed model of the original problem. If a value of x_i is set to 0, then the value of its corresponding variable s_i will always be greater than or equal to 0, which will result in reducing the dual value of task i in equation (3.5). Consequently, in the next iteration, either the value of x_i will become 1, or the number of assigned time units to this task will get smaller.

On the other hand, if the value of x_i is set to 1, then based on the number of time units assigned to the task, s_i will be positive or negative. If positive, then task i will get time units more than what it requires, which causes equation (3.5) to reduce the dual value of the task. And the other way around in case when it is negative.

Notice that the cost-effectiveness ratio of a task is $\frac{w_i}{p_i}$. Based on this ratio, we can decompose the values that a dual variable can take into three intervals, namely:

1. $\lambda_i > \frac{w_i}{p_i}$: In this interval, x_i will always be 0, and at the same time, it is possible to assign time units to the task since the dual variable is relatively large. The gradient descent, in this case, will keep reducing the dual variable until Procedure 2 assigns 1 to x_i as λ_i entered the second interval. Or it assigns no time units for the task, which means it decided not to schedule this task unless the dual variables of other tasks decrease sufficiently.

2. $0 \leq \lambda_i \leq \frac{w_i}{p_i}$: x_i will always be 1. But, this doesn't mean gaining the entire task weight. A portion of the weight is given to $\{y_{it}\}$, and to get this portion back, p_i of these variables must be set to 1.

3. $\lambda_i < 0$: x_i will always be 1, but at the same time, none of $\{y_{it}\}$ will ever be 1. This will make the gradient descent algorithm keeps increasing the dual variable until it enters the second interval (i.e., $0 \leq \lambda_i \leq \frac{w_i}{p_i}$).

Based on the above discussion, it becomes clear that the typical situation is when the dual variable lies on the second interval. In this interval, the gradient descent keeps fluctuating (increasing/decreasing) the dual variable until it finds a proper value which makes Procedure 2 assigns enough time units to the task (but not overly high). If increasing the task variable does not change the situation, the dual variable will enter the first interval in which it might get rejected. Therefore, we decided to let the initial dual values be $\frac{w_i}{2p_i}$.

3.5.1.3 Attaining the optimal or near optimal solution

Notice here that the method may not attain the optimal (or near-optimal) solution of the original problem directly by solving the Lagrangian dual problem for most instances. We mentioned earlier that the standard situation is when the value of the dual variable is between 0 and the cost-effectiveness ratio of the task (i.e., $0 \leq \lambda_i \leq \frac{w_i}{p_i}$). In this interval, Procedure 2 will not stop changing the value of the dual variable unless its corresponding task is assigned with exactly p_i time units. However, since the lifespan of a task is usually larger than its processing time, then most of the time, more than p_i time units are allocated for task i. Consequently, the sub-gradient descent keeps changing the dual value of task i without converging. In addition, the extra time units given to this task will probably block other tasks from being scheduled. Note that even if we remove the extra time units assigned to a task in the

feasible solution obtained from the Lagrangian relaxation, it will not help schedule more tasks. Because, in the feasible solution, we do not schedule tasks; we only remove the infeasible ones. On the other hand, removing the tasks assigned with time units more than they needed will empty the scheduler. To overcome this problem, we limit the number of time units assigned to a task by p_i. We refer to this method as the limited integrality constraint Lagrangian (LICL). Now, we have a better chance to reach the optimal (or near-optimal) solution.

Proposition 3.4. *LICL provides a better bound from the linear programming relaxation.*

Proof. The integrality constraint (C2) can be represented by these two constraints:

$$\sum_{t \leq T} y_{it} \geq p_i x_i \quad \forall i \in \mathcal{I} \tag{3.6}$$

$$\sum_{t \leq T} y_{it} \leq p_i x_i \quad \forall i \in \mathcal{I} \tag{3.7}$$

Now, by relaxing only constraint (3.6), the second constraint (3.7) ensures the limitation mentioned in this section (i.e., limiting the number of time units assigned to task i by p_i). Hereupon, the matrix entries of the LICI's constraints are not within the values of 1, 0, or -1 (due to the existence of p_i in constraint (3.7)). Therefore, the matrix is not totally unimodular. Hence, the feasible region obtained from this relaxed model is not the convex hull of the relaxed problem. Thus, we conclude that LICL does not have the integrality property, and so it provides a better upper bound than the standard linear programming relaxation. □

Proposition 3.5. *For every instance of the original problem, there is at least one dual value ($\boldsymbol{\lambda}$) that can result in an optimal feasible solution.*

Proof. First, we have to prove that for any feasible solution A, there is a dual value(s)

that can generate a feasible solution B, which is identical or better than A. We prove this by induction.

For the base case, let us assume, at time unit $t = 1$, a number of tasks are scheduled in the feasible solution A. Then, we set their dual variables to the highest possible value so as to be scheduled by Procedure 2. If there are some vacant machines in this time unit, algorithm 2 will assign extra tasks to these machines that satisfy all the constraints. Note that in the feasible solution construction, we will remove any incomplete tasks. So far, in the base case, solution B is at least as good as A.

For the inductive step, let us assume, at time unit $t = k$, solution B is at least as good as solution A. Now, at time unit $t = k + 1$, tasks that already allocated enough time units for their processing time will not be considered for the current time unit since we limited the time units given to task i by its processing time p_i. If any task is out of its lifespan, it won't be considered for this time unit as well. If there are tasks in the previous time units that have not been assigned enough (i.e., the number of time units assigned to task i is less than p_i), either they will be assigned the current time unit or kept for future time units. This decision will be based on the value of their dual variable. Moreover, Procedure 2 will fill empty machines (if any) with tasks that have the highest dual values. Consequently, after constructing a feasible solution from t units, we end up with a solution (B) which is better, or in the worst case, similar to A (i.e., the number of scheduled tasks in B is greater than or equal to A). Up to here, we showed that $Objective(B) \geq Objective(A)$. Now, assume that we are given the optimal solution. We will follow the algorithm explained above to get the dual values and schedule the tasks using algorithm 2. To construct the feasible solution, We remove tasks that have not been scheduled completely, and since the solution is optimal, there will be no tasks except the ones shown in the optimal solution, or otherwise, it will contradict the optimality of the solution. □

3.5.2 The Capacity Constraint Lagrangian Relaxation (CCL)

Here, in this subsection, we relax constraint C3 (capacity constraint). The following is the Lagrangian relaxation model of the capacity constraint while rescheduling the pre-admitted tasks:

$$
\begin{aligned}
L_{c3}(\boldsymbol{\mu}) = \Big(& maximize \sum_{i \in \mathcal{I}} w_i x_i \\
& + \sum_{t \leq T} \sum_{i \in \mathcal{I} \cup \mathcal{I}'} \mu_t y_{it} \\
& - \sum_{t \leq T} \mu_t M \\
s.t. & \\
y_{it} = 0 \quad \forall i, t \quad & : \quad t \geq min(\hat{\alpha}_i, \beta_i) \\
& \vee \quad t \leq \gamma_i \\
\sum_{t \leq T} y_{it} = p_i x_i \quad & \forall i \in \mathcal{I} \\
\sum_{t \leq T} y_{it} = p_i \quad & \forall i \in \mathcal{I}' \\
x_i \in \{0,1\} \quad \forall i \quad y_{it} \in \{0,1\} \quad & \forall i, t \Big)
\end{aligned}
$$

We propose algorithm 1 to solve the Lagrangian relaxation of the problem $L_{c3}(\boldsymbol{\mu})$. Relaxing the capacity constraint (C3) removes the constraint on the number of machines; the number of tasks assigned in a single time slot is not restricted to the number of available machines. Thus, it is possible to schedule any number of tasks to improve the objective function. Notice that the value of the dual variables in this relaxation model is negative. If a task is decided to be scheduled, based on constraint

C2, enough processing time units (i.e., p_i time units) related to machines should be assigned to the task (task i). Therefore, to decide whether to schedule a task or not, Procedure 1 first sorts in descending order the dual values corresponding to the available time units of tasks' life span (line 5). Next, it sums up the first p_i dual values in the sorted list (lines 6-8). Then, the sum is added to the task's weight. Finally, if the result is positive, Procedure 1 schedules the task. Otherwise, it rejects the task (lines 9-12).

Algorithm 3 Solution for the relaxed ICL problem

```
1:  procedure CCL-UPPER-BOUND(μ, w, p, α, β, γ)
2:      x_i ← 0  ∀i ∈ I
3:      y_it ← 0 ∀i ∈ I ∪ I'   ∀t ≤ T
4:      for i ∈ I ∪ I' do
5:          t = sort(γ_i, min(β_i, α_i), μ)
6:          sum ← 0
7:          for u = 1 to p_i do
8:              sum = sum + μ_it(u)
9:          if w_i + sum > 0 ∨ i ∈ I' then
10:             x_i ← 1
11:             for u = 1 to p_i do
12:                 y_it(u) = 1
        return {x, Y}
```

It is easy to show that Procedure 3 returns the optimal solution of the relaxed model $CCL(\lambda)$; we can prove this similarly to how we proved proposition 3.2.

Further, the time complexity of the CCL algorithm is $O(\hat{N}Tlog(T))$; we can show this in a similar way to that of algorithm 2.

3.5.2.1 Obtaining a feasible solution for the CCL problem

To obtain a feasible solution, after limiting the number of machines, we have to satisfy two conditions: 1) all the pre-admitted tasks are scheduled, and 2) the number of time units assigned to each task is not less than its processing time. For the first condition, it is enough to check, at each time slot, the number of assigned pre-admitted tasks is

not greater than the number of machines. As for the second condition, first, we sort all the scheduled tasks in descending order based on their updated weights calculated in line 10 of Procedure 3. Then, for every element in the sorted list, we check its assigned time units whether they are all within the number of machines. If yes, the task is accepted; otherwise, it is rejected.

3.5.2.2 Updating dual variables using sub-gradient descent

Let the vector $Z =< z_1, z_2, ..., z_T >$, where $z_t = \sum_{i \in \mathcal{I} \cup \mathcal{I}'} y_{it} - M$. Then the dual variables $\{\boldsymbol{\mu}\}$ are updated using the following equation:

$$\boldsymbol{\mu}^{\tau+1} = \boldsymbol{\mu}^{\tau} + \frac{\epsilon * [FS(\boldsymbol{\mu}^{\tau}) - L_{c3}(\boldsymbol{\mu}^{\tau})]}{\|Z\|^2} Z \tag{3.8}$$

The intuition behind this equation is straightforward. When the number of tasks assigned to machines, in time slot t, is greater than the number of available machines, then the sign of z_t is positive, which causes the dual value of t in the next iteration to decrease. Alternatively, if the number of tasks is lower than the number of machines, then the sign of z_t is negative, which causes the dual value to increase in the next iteration. The dual variables in this Lagrangian relaxation method represent the priority of time units used in scheduling tasks. When a time unit is overwhelmed with tasks in the current iteration, then, in the next iteration, a lower priority is given to the task, and vice versa. The sub-gradient descent method will keep iterating until it converges.

3.5.2.3 Optimality

In ICL, we improve the expected outcome by adding limitations to the algorithm, solving the dual Lagrangian problem. However, in CCL, no limitation can help in reaching the optimal solution. The only possible restriction that we can address here

is to let the number of assigned tasks to each time unit not be greater than the number of machines. However, this limitation makes the performance of the method even much worse than before. Because it reduces the number of scheduled tasks in any infeasible solution, which, in turn, limits our choices for extracting a feasible one.

We can prove that CCL provides a better bound than the linear program relaxation the same way we did for ICL, but we omitted the proof for the purpose of brevity.

We found that proving the optimality of CCL is a challenging task. An example that shows the difficulty is the following: Consider a scenario where we have ten tasks, each with a fixed processing time of one (i.e., $p_i = 1$, $i = 1, 2, ..., 10$), a lifespan of two, $T = 2$, and a total of five machines. Now, if we set the dual value of the first time unit higher than the second time unit, then the method will schedule all the ten tasks in the first time unit. Similarly, if the second time unit has a higher dual value, all the ten tasks will be scheduled in the second time unit. If both have the same dual value, tasks will be assigned to any time units without restriction. Meanwhile, constructing a feasible solution from the given solutions, half of the tasks will be removed. However, the optimal solution is when all tasks have been scheduled (five tasks in each time unit).

3.6 Greedy algorithm

As explained earlier, the problem of scheduling the maximum number of tasks at a vehicular edge cloud is NP-hard. To overcome the complexity, we have solved the problem using Lagrangian relaxation. However, the method requires an initial feasible solution to boost its performance. Hence, in this section, we present a scalable and efficient algorithmic way, where at each scheduling epoch, the algorithm, after scheduling and allocating time slots for all previously admitted tasks, greedily assigns

a maximum number of new arrival tasks on available time slots. The steps of the greedy algorithm are shown in algorithm 1.

Algorithm 4 A greedy heuristic to solve the problem

```
1: procedure GreedyAlgorithm(I', I))
2:     Phase I: I'_Sort ← sort(I', s.t. α_i ≥ α_{i+1}) F_m = T  ∀m ∈ M
3:     for i ∈ I' do
4:         B ← max(F_{m∈M})
5:         t = min(F_B, α_i)
6:         for j = 1 to p_i do
7:             Slot_t^B ← i
8:             t = t − 1
9:         F_B = t
10:    Phase II:
11:    I_Sort ← sort(I, s.t. α_i ≤ α_{i+1})
12:    m = 1
13:    for i ∈ I do FindSlot ← True
14:        while FindSlot do
15:            t = γ_i
16:            if Slot_t^m == ∅ then
17:                t' = t
18:                Empty ← True
19:                for j = 2 to p_i do
20:                    t' = t' + 1
21:                    if Slot_{t'}^m != ∅ || t' > α_i then
22:                        Empty ← False
23:                if Empty then
24:                    for j = 1 to p_i do
25:                        Slot_t^m ← i
26:                        t = t + 1
27:                    FindSlot ← False
28:            if FindSlot then
29:                if t < α_i then
30:                    t = t + 1
31:                else if m < M then
32:                    m = m + 1
33:                    t = γ_i
34:                else
35:                    FindSlot ← False
```

The algorithm has two phases. In Phase I (Lines 2-10), all the admitted tasks in

the set $\mathcal{I}'$ are allocated to time slots. In more details, first, the algorithm sorts the set $\mathcal{I}'$ in descending order based on the tasks' deadline (i.e., each item i in the sorted list $\mathcal{I}'_{Sort}$ has a time deadline α_i greater than or equal to α_{i+1}, Line 2). Next, for each admitted task i, the algorithm allocates the farthest sequential empty time slots available for the set of machines that guarantee the deadline α_i. To do that, initially, the algorithm for all machines (i.e., $\forall m \in \mathcal{M}$) assigns the farthest empty time slot (i.e., $\mathcal{F}_m$) to the last available time slot (i.e., $\mathcal{F}_m = T$, Line 3). Next, the algorithm for each admitted task i in the set $\mathcal{I}'$, chooses the emptiest machine (i.e., machine with farthest empty slot, $max(\mathcal{F}_{m\in\mathcal{M}})$, Line 5), and allocates sequential p_i time units for the chosen machine $\mathcal{B}$, ending with $min(\mathcal{F}_{\mathcal{B}}, \alpha_i)$, to guarantee the task's deadline (Lines 7-9). In Phase II (Lines 11-37), a maximum number of newly arrived tasks is greedily chosen and assigned to available time slots. Initially, the set of arrived tasks $\mathcal{I}$ is sorted in ascending order based on tasks' deadline (Line 12). Next, from the first machine (i.e., $m = 1$), for each task i in the sorted set $\mathcal{I}_{Sort}$, the algorithm allocates the earliest empty time slots that meet the release time γ_i and the deadline α_i. If no such time slots are found in the current machine, the algorithm tries the next machines until they fit or drop the task. The While loop in Line 16 ensures that the algorithm will keep searching for suitable time slots for task i while the flag $FindSlot$ is $True$. At each iteration, the time t is initially set to the task's release time γ_i (Line 17). If the time slot for machine m and time t is empty (

Line 18), the algorithm checks whether a sequence of p_i units is empty (Lines 19-24). If yes, it allocates those time slots for task i and sets the flag $FindSlot$ to $False$, meaning the task has been scheduled (Lines 25-29). Otherwise, when $FindSlot$ is $True$ (Line 30), if the current time t is less than the task's deadline, the time t is incremented (Lines 31-32), else if there are unvisited machines, the next machine is assigned, and time t is reset to γ_i (Lines 33-35), else the task is dropped (Lines 36-37).

3.7 Performance Evaluation

We study here the performance of the proposed methods, including Integer Linear Programming (ILP) (to get the optimal solution), Integrality Constraint Lagrangian (ICL), Capacity Constraint Lagrangian (CCL), and greedy algorithm (Greedy). We consider different performance metrics such as execution time, performance gap, and weighted rejection rate. We then study the overall weighted rejection rate for the overall system using the ICL method by varying the capacity (number of computational VMs) and arrival rate of the tasks. Finally, we analyze the performance of the system in terms of the rejection rate. Vehicular traffic traces are obtained using the well-known traffic simulator SUMO. To imitate a realistic scenario, we chose the vehicle's emitting probability (per second) as one. The vehicle's speed follows a truncated Gaussian distribution with mean equals 50 KM/h and variance equals 40.

For comparison purposes, we use two different strategies: 1) *Rescheduling-Pre-Admitted-Coding* (RPA-Coding); that is, rescheduling the pre-admitted tasks along with the new arrival tasks at each scheduling epoch. 2) *Fixing-Pre-Admitted-Coding* (FPA-Coding); that is fixing the time slots which have been allocated for pre-admitted tasks and scheduling only the newly arrival tasks on available time slots.

We assume an RSU with 2×500m coverage range, equipped with 20 VMs to process offloaded workloads. The wireless channel bandwidth is assumed to be B = 300MHz and assume SNR values to cover a range of data rates between 1Mbps (the farthest from the RSU) and 400Mbps (closest to RSU); here $SNR_s = 2^{r_s/B} - 1$. Other simulation parameters are shown in Table 5.2. We use CPLEX to solve our optimization models and C++ to simulate the operation of our algorithms through a discrete event-driven simulation. We generate results on CPU with Intel(R) Core(TM) i7-6700 CPU @ 2.7GHz, 16GB memory ram, and 64-bit mac operating system. The results are averaged over ten runs.

Table 3.1: Measurable factors used for the scheduling performance

Factors	Distribution	Mean	Variance
Task Arrival Rate (tasks/second)	Exponential	200	-
vehicle's Velocity (KM/H)	Trunc. Gaussian	50	40
Delay intolerant deadline (s)	Gaussian	0.1	0.03
Delay tolerant deadline (s)	Gaussian	1	0.1
Upload Data Size (KB)	Gaussian	500	1
Download Data Size (KB)	Gaussian	100	1
Weight given for the task	Gaussian	5	3
Delay Intolerant Processing Time (s)	Gaussian	0.02	0.01
Delay Tolerant Processing Time (s)	Gaussian	0.5	0.1

3.7.1 Scheduling performance

This section evaluates the scheduling performance of the different proposed methods; the obtained results are depicted in Fig. 3.4, and Table 3.2. Here, we first assume a fixed number of vehicles within the range of the RSU and vary the number of tasks (50-200) generated by the vehicles to test the scalability and quality of the workload scheduling methods. Our comparison metric here is the rejection rate of requests; the tasks are generated such that the processing time of each can fit within its lifespan.

As shown in 3.4, as the number of tasks increases, the run time of the ILP exponentially increases. When the number of tasks is 50, the ILP takes around 5 seconds, and the ICL, CCL, and Greedy take 0.35s, 1.344s, and 0.1134s, respectively. However, for 200 tasks, the ILP takes around 538s, while the ICL, CCL, and Greedy take 3.58s, 12.34s, and 0.558s, respectively. The latter three methods all exhibit polynomial run time in the size of the problem, whereas the ILP, suffers from the exponential run time. The Greedy method enjoys the fastest run time indeed, as expected. However, Greedy yields solutions that are far from optimal, as indicated in Table 3.2.

The table shows rejection rates obtained by the various methods and the gap of each of the methods from the optimal solution. First, the table shows that as the number of offloaded tasks increases, the system will start rejecting some of the tasks before scheduling them on the VEC resources. Indeed, initially, as the

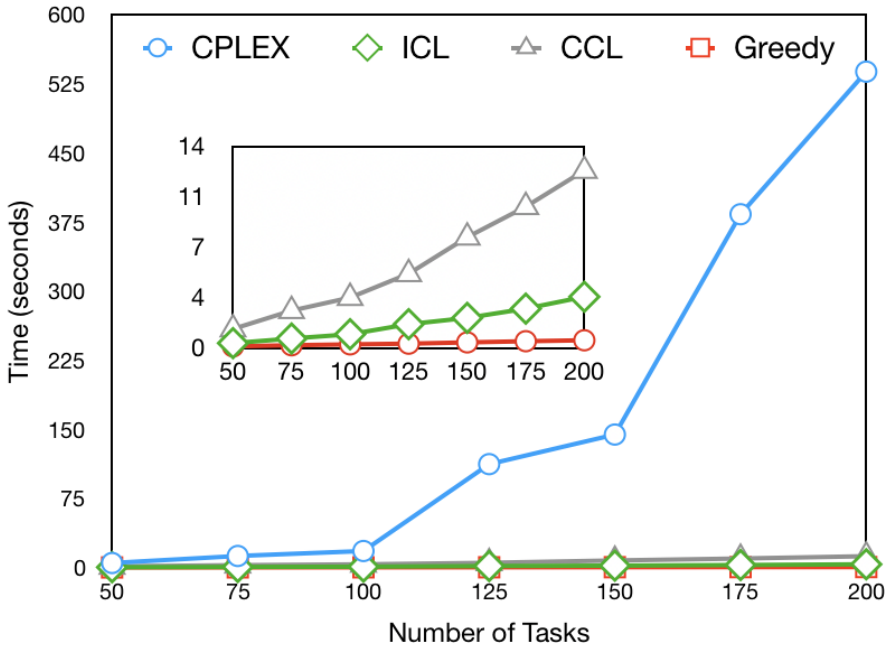


Figure 3.4: Computational complexity of different methods.

cars offload their tasks, the system schedules them onto the available machines. Given the fixed resources (e.g., CPU cycles) per machine, the more tasks are offloaded, the higher the response times are going to be. Therefore it becomes difficult to admit them and complete them before their deadlines, and consequently, the system rejects those tasks that cannot meet their deadlines. Here, the ILP allocates and optimally schedules the offloaded workload and therefore achieves at all times the lowest rejection rate. Whereas, for the other methods whose solutions are only suboptimal, they exhibit a drift from the optimal solution, which is shown as a deviation in Table 3.2. Rejection rate $= \frac{\text{weighted num. of admitted tasks}}{\text{total weight}}$, and deviation $= \frac{\text{optimal obj. - weighted num. of admitted tasks}}{\text{optimal obj.}}$.

Now, we should note here that both ICL and CCL are iterative methods. As we increase the number of iterations, one expects to improve the quality of the solution, which comes at the expense of higher computation times. To validate this, we study the performance (objective being the weighted sum of admitted tasks) of an instance

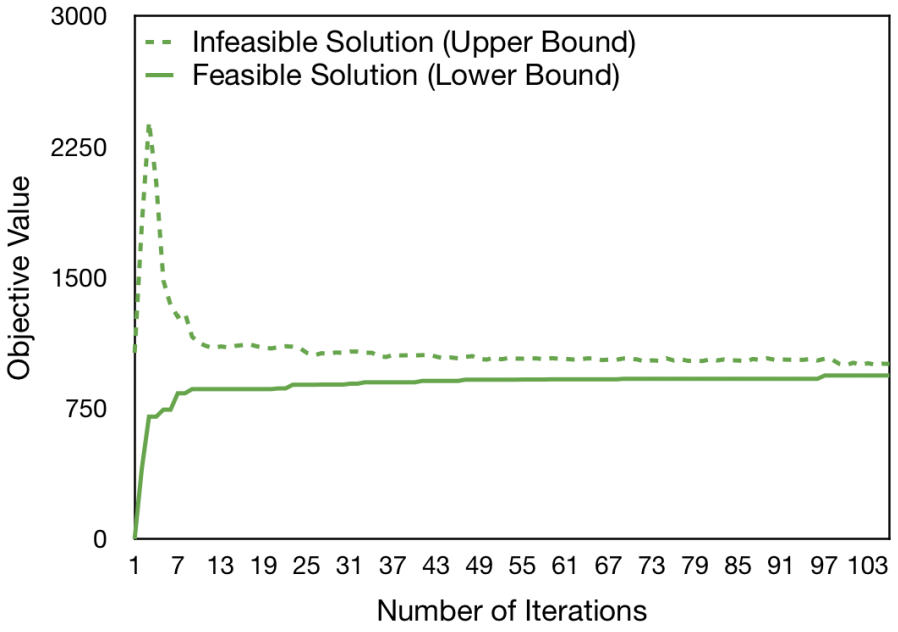


Figure 3.5: The convergence of ICL towards the optimal solution

of ICL (with 200 tasks), as we iterate towards improving the solution (see Figure 3.5. We show both the (infeasible) upper and (feasible) lower bounds on the same figure; the upper bound and lower bounds both converge towards a solution, where after 100 iterations (approx. 3.5s), the gap between the two is reduced to 7%. Given the superior performance of ICL (over CCL and Greedy) and the slow run time of the ILP, in the rest of our evaluation, we only consider the ICL method.

From the study and analysis conducted in this section, we conclude that the ICL method is the best choice for task scheduling in terms of both execution time and minimizing the rejection rate of the tasks. Hence, in the next section, to evaluate the performance of our system simulation, we use this method for the scheduling stage.

3.7.2 System Simulation

This section implements an event-driven simulation to simulate the behavior of a vehicular network with the dynamic arrival of cars and the dynamic generation of

Table 3.2: Scheduling Performance.

number of tasks	ILP Rej. Rate	ICL Rej. Rate	ICL Dev.	CCL Rej. Rate	CCL Dev.	Greedy Rej. Rate	Greedy Dev.
50	6%	13%	7%	26%	21%	26%	22%
75	9%	16%	7%	31%	24%	34%	28%
100	13%	19%	7%	31%	21%	37%	28%
125	14%	22%	9%	33%	22%	39%	28%
150	14%	22%	9%	34%	24%	39%	29%
175	14%	22%	9%	34%	23%	40%	30%
200	14%	22%	9%	33%	22%	40%	30%

tasks. The vehicles' arrival process is simulated through the realistic vehicles traffic simulator SUMO . We simulate a 1KM bi-directional road segment with three lanes for each direction. We assume the RSU is in the middle of this road segment and the perpendicular distance between the road and the RSU is 20Meters. We assumed the following values for the simulation: buffering time: 0.05Seconds, scheduling time: 0.05Seconds, Sumo vehicle arrival probability per second: 1, and tasks request rate per second: 20. The rate of requested tasks per vehicle is assumed to follow a Poisson process, and the inter-arrival time between two consecutive requests follows an exponential distribution. As explained earlier, we consider two classes of tasks: the time-sensitive (60%) and high bandwidth tasks (40%). The ICL method is the one used for scheduling the workload. We simulate the system for a total of 5 seconds. Here, at each scheduling epoch (5ms), the system gathers the requests to offload the tasks and invoke the ICL to schedule the workloads (each time with and without the ones already waiting in the system, RPA, and FPA, respectively).

We consider two types of events: request arrival and scheduling events. The events are generated as follows: We first run SUMO and monitor the arrival of vehicles at every time unit. If a vehicle arrives, we generate the tasks requests for this vehicle up to the maximum simulation time and save these events to a sorted data structure. If a

vehicle leaves the road segment, we update its departure time. We go through all the vehicles and their generated events. If an event occurred between the previous and current units, we would update the event location with the vehicle's current location. After going through the simulation duration, we remove all the events that occurred after the vehicle departure time and then add scheduling events to the events data structure. We assume that the maximum legal speed of this road is 80 km/h and that 95% of the vehicles run at speed in the range 80% to 120% of the legal speed limit. The maximum acceleration is 3.0 km/h^2 and maximum deceleration is 6.0 km/h^2. The speeds of the vehicles follow a gaussian distribution with mean equals the road's legal maximum speed, and the variance is 0.1.

Fig. 3.6 evaluates the system performance of the ICL method by varying the number of computational VMs. We fix the request arrival rate to 20 req./s. For comparison purposes, we use RPA-Coding and FPA-Coding. As shown in the figure, it is clear that as we increase the number of computational machines, the overall weighted rejection rate for both coding methods decreases linearly. For instance, when the number of VMs is 20, the weighted rejection rate of RPA-coding (FPA-coding) is 17.5% (19.5%), and when the number of VMs is 80, the rejection rate is 6% (10%). This even demonstrates that the gap between the rejection rates of the two coding methods slightly increases as the number of VMs increases. Recall that the RPA-coding has more flexibility because it can reschedule the pre-admitted tasks. For the same reason, as illustrated in the figure, RPA-Coding consistently outperforms FPA-Coding. Most importantly, from this study, we can analyze the capacity of the MEC resources to be provisioned for acceptable quality of service.

In Fig. 3.7, we study the system performance in terms of weighted rejection rate by varying the arrival rate from 10 requests per second to 25. In this study, we fix the number of computational VMs to 20. The figure shows that the RPA-Coding performs

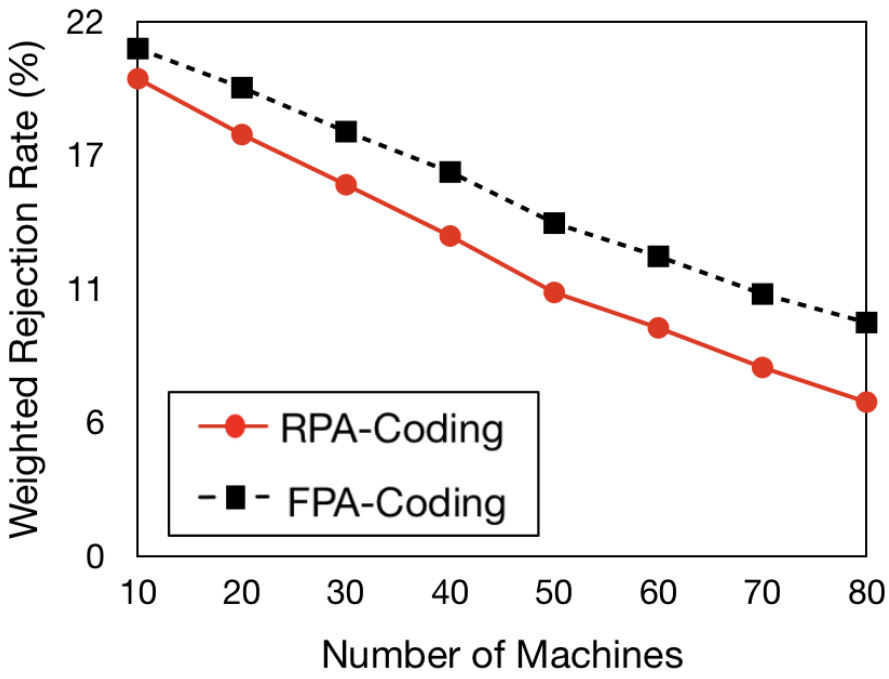


Figure 3.6: Rejection rate Vs number of machines.

better than the other coding method with at least 3% lower rejection rate. Notice that such gain is attributed to the flexibility of rescheduling the pre-admitted tasks. However, this gain comes at the expense of additional complexity and computational time. From the figure, we may also observe that the rejection rate of both coding methods gradually increases with the increase of the requested arrival rate. Indeed, when the number of arrival tasks in the system increases, the system starts to reject more tasks depending on processing capability and tasks' deadline. However, the figure shows that even at the load of 25 req./s, the rejection rate is still acceptable.

Fig. 3.8 plots the rejection rate of the two ICL coding methods by varying the percentage of delay-tolerant tasks to delay intolerant. This study will help us determine the type of tasks in terms of their latency the VEC system should admit in the pre-processing stage to increase the performance. The computational capacity of the system is set to 20 VMs, and the task arrival rate is fixed to 20 tasks per second. The figure shows that when the percentage of tolerant to intolerant tasks is low (for

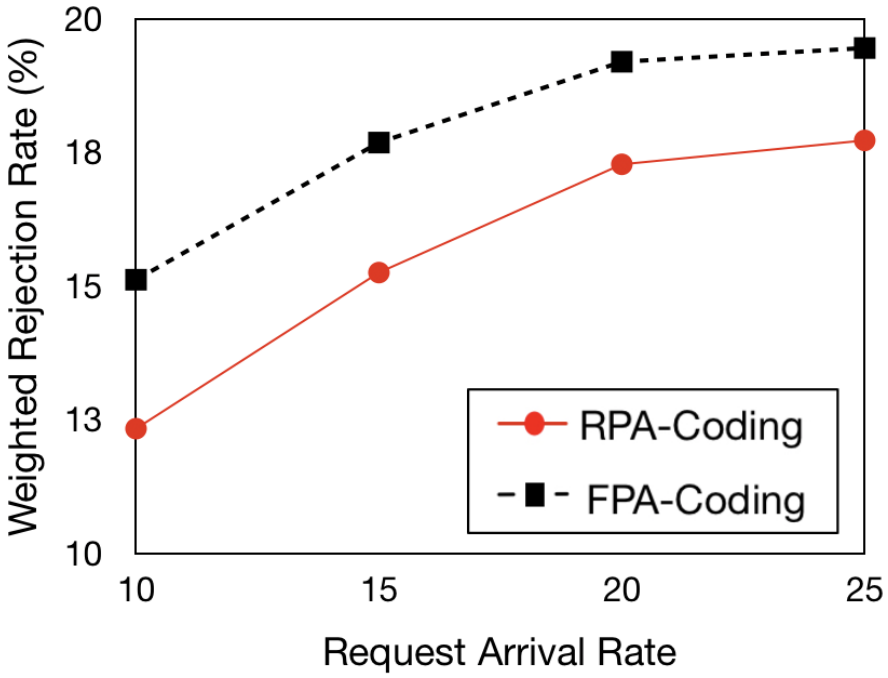


Figure 3.7: Rejection rate VS request arrival rate.

instance, 25%), the rejection rate is very low (respectively 5% for RPA-Coding and 8% for FPA-Coding). Also, when the percentage of tolerant to intolerant tasks increases (for example, reaches 75%), the rejection rate increases accordingly (res. the rejection rate of RPA-Coding and FPA-Coding becomes 21% and 22%). The reason can be explained as follows: when most of the tasks to be scheduled in the system are intolerant tasks, due to their small sizes and short processing time requirement, the scheduler can easily adjust them at available time slots, and thus accepts more tasks. Whereas, when the number of tolerant tasks to intolerant ones is more significant, the scheduler faces difficulty fitting them on available time slots. The scheduler will mostly accept the intolerant tasks and rejects the tolerant ones. Hence, to balance the two types of delay tasks, the smart RSU is crucial. This study is out of the scope of this work and is kept for future work.

Figure 3.9 shows the performance of the system simulation for five seconds. The figure depicts the fluctuation of the rejection rate for the two coding strategies within

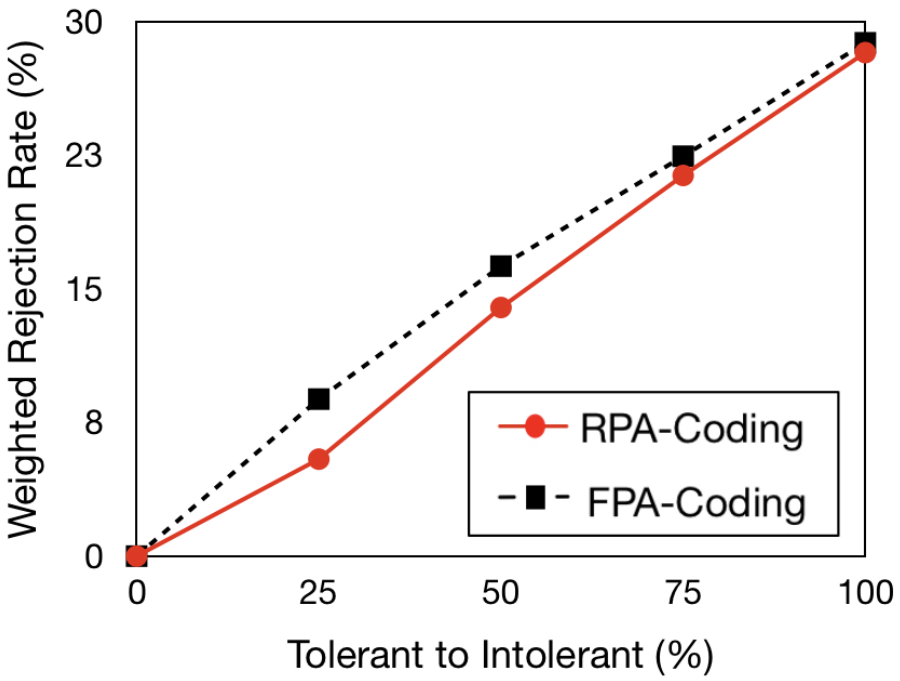


Figure 3.8: Delay-tolerant to delay-intolerant tasks ratio.

these seconds. For the first second, the performance of both RPA-coding and FPA-coding schemes are almost the same. After that, as the machines of the systems started to get filled up, FPA-coding finds difficulty to get free time slots for the new tasks. That is because this coding method does not reschedule the pre-admitted tasks. On the contrary, the RPA-coding, because of rescheduling the pre-admitted tasks, makes more room for new arrival tasks.

In addition to the rejection rate, we study the system's performance in terms of tasks completion time. Indeed, evaluating the absolute completion time is crucial if the objective is to reduce the task's latency. However, since in this work, the ultimate goal is to increase the weighted number of admitted tasks, where each task has its completion deadline, hence, studying the *earliness* of the accepted tasks will give better insights. The earliness for each task (say task i) is computed as follows:

$$earliness(i) = w_i \frac{\alpha_i - \phi_i}{\alpha_i - \psi_i} \tag{3.9}$$

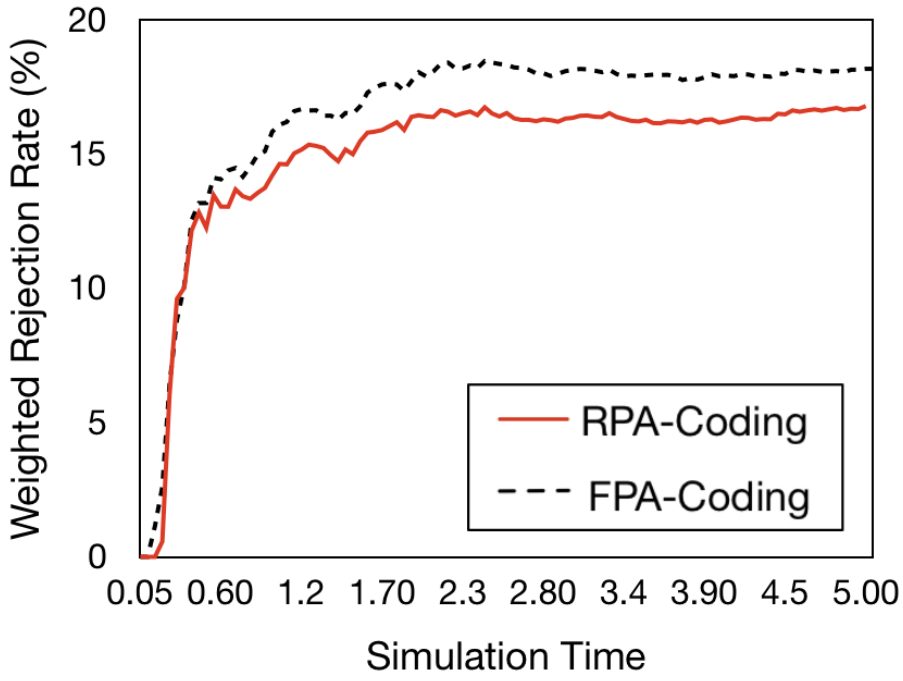


Figure 3.9: Rejection rate VS simulation time

where ϕ_i and ψ_i are respectively the completion time and the request time of task i.

Fig. 3.10 evaluates the effectiveness of the system as a function of earliness by varying the number of computational VMs. The request arrival rate is set to 20 req/s, and the number of machines is varied from 10 to 70. As shown in the figure, regardless of the number of VMs, it is clear that due to the rescheduling capability, the RPA-Coding schedules tasks faster than FPA-Coding. We may also infer that the earliness of both methods increases with the increase of the number of VMs. However, the RPA-Coding grows more quickly than the other, since the former has more flexibility to schedule tasks with early deadlines first, without being affected by pre-admitted tasks. Hence, as can be seen, the gap between the two coding schemes increases with the number of VMs. For instance, with 70 VMs, the gap is almost twice the gap with 30 VMs.

In Fig. 3.11, we illustrate the effectiveness of the system by varying the requested arrival rate from 10 to 25 req./s. In this evaluation, two system sizes, i.e., 20 and

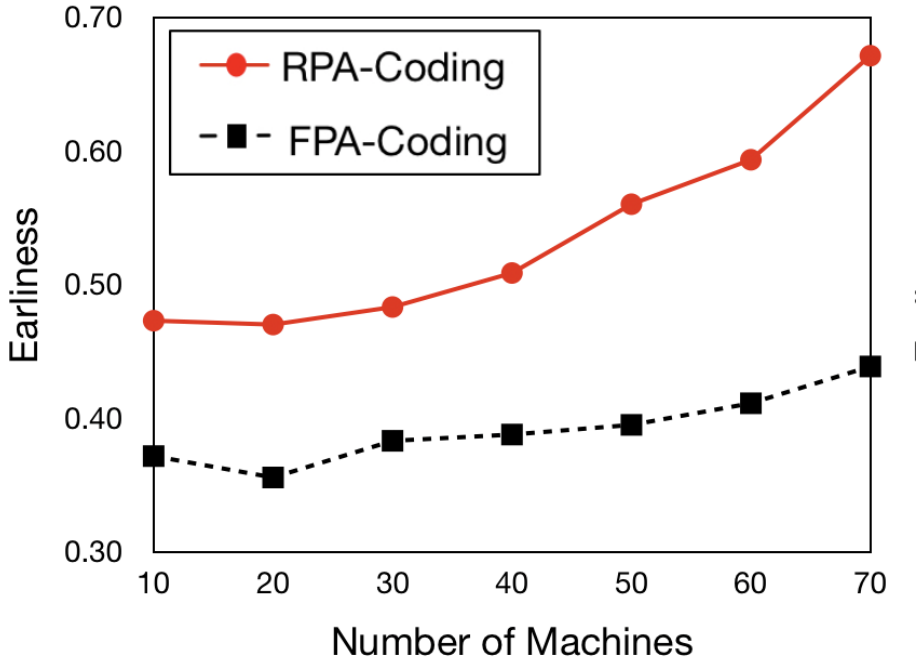


Figure 3.10: Earliness VS no. of machines.

60 VMs, are considered. In fact, with 60 VMs, when the arrival rate is very low (for example, ten req/s), the system can schedule all the requested tasks without being filled. As shown in the figure, tasks are being scheduled much earlier than their deadlines. Hence the earliness is shown to be very high. Clearly, as the arrival rate increases, the system occupies more tasks, and thus the earliness starts to decrease. Whereas, in a system with 20 VMs, when the arrival rate is very low, because of the small number of VMs, all the computational resources will be occupied, and therefore the earliness is shown to be low. By increasing the arrival rate, the earliness does not show much difference. That is because all the system's computational resources are fully occupied and have no room for new tasks. Hence, along with the arrival of more tasks, the system starts to drop the overwhelmed tasks, and thus the earliness stays unchanged.

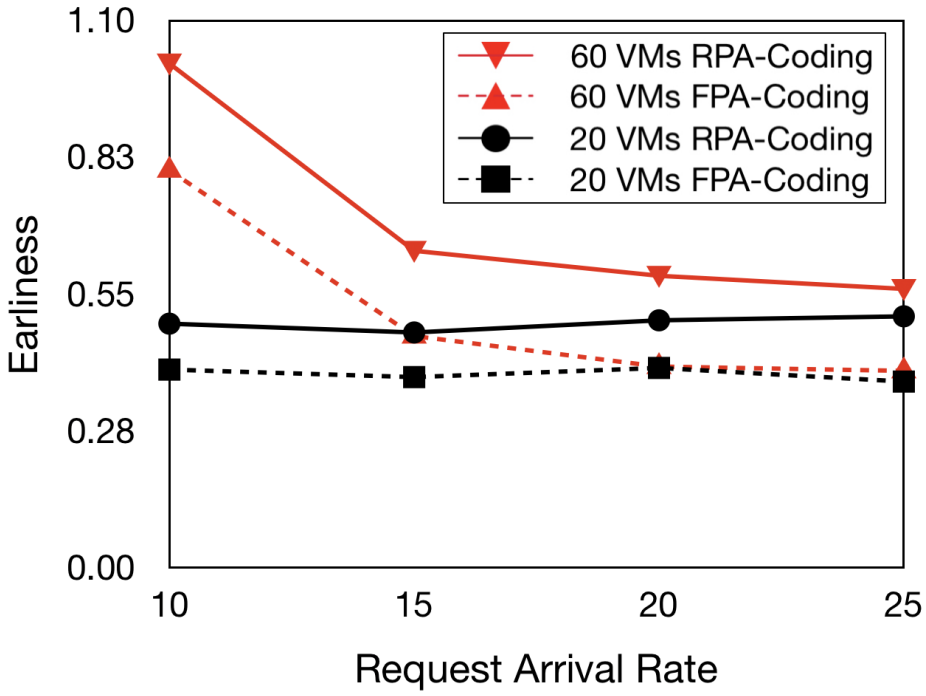


Figure 3.11: Earliness VS request arrival rate.

3.8 Conclusion

We considered a vehicular cloud edge system that brings capabilities to deploy data storage, computing, and communication and a host of support applications for emerging vehicular services to the network edge, closer to where the users are, thereby reducing response times and saving core bandwidth. However, a typical challenge for VEC is related to computation offloading which should consider the high mobility of vehicular networks. In addition, various offloaded tasks may have different resource requirements (e.g., computation resources for task execution, communication resources, etc.). We studied the problem of workload offloading and scheduling at VEC resources deployed alongside a roadside unit and providing vehicular services to driving-by cars. We consider pre-processing tasks before admitting them and then solve a scheduling problem for all admitted tasks to meet their deadlines. We showed that the scheduling problem is NP-hard and present polynomial-time solutions

based on Lagrangian relaxations to attain efficient solutions. Our obtained results are compared with the optimal solutions. Our findings are beneficial for dimensioning vehicular cloud edge systems in terms of capacity and resources, considering the high mobility and density of vehicles and the intensity of offloaded tasks per vehicle.

Chapter 4

An Infrastructure-Assisted Workload Scheduling for Computational Resources Exploitation in Fog-Enabled Vehicular Network[1]

[1]This chapter has been published in IEEE Internet of Things Journal [74].

As we concluded in chapter three, the validity of deploying and utilizing MEC in a vehicular network is now apparent. This chapter accumulates over the previous one with two components: First, it considers limited wireless resources available hence promoting the model feasibility. Second, it utilizes idle resources in the vehicles' OBUs as an expansion to the infrastructure computational resources. Fog computing is a framework that facilitates the exploitation of computational resources available at the very edge of the network (e.g., users' equipment, vehicles, etc.). By utilizing idle resources at that level, we can expand our system beyond using the infrastructure resources to exploiting the computational potential of any network edge node. Such a framework will reduce the end-to-end communication delay and increase the service's reliability while reducing the amount of bandwidth consumed in the backhaul network. VeFC broaches the concept of VaaR by harvesting the vehicles' OBU computational capabilities. In this work, we propose a system deployed on RSUs, to collect low-latency computational tasks offloading requests from different sources and efficiently schedule them on available OBUs before exiting the RSUs' coverage range.

4.1 Motivation

In recent years, the conventional solution of inflating computational processing demands in vehicular networks was to offload the intensive tasks over the distant cloud servers through the RSU. Such an RSU-to-Cloud communication exhibits considerable delays that marginalize the cloud servers' efficiency for handling delay-sensitive services, including significant bandwidth consumption when forwarding tasks in periods of elevated offered loads. Under such circumstances, MEC presents itself as an alternative setting that yields lower end-to-end delays through the deployment of computationally-capable servers (*a.k.a.* cloudlets) at the network's edge close to the RSU (*e.g.* [75]). However, these MEC servers inevitably find themselves unable to

cope with high offered loads due to the timely unavailability of adequate computational resources; this being a limitation that obliterates MECs' short-latency virtue (*e.g.* [18]). An early solution suggested in [18, 19, 76] was to organize cloudlets in a hierarchy of multiple edge-tiers. With this arrangement, lower-tier cloudlets are allowed to issue task migration requests to upper cloudlet tiers. Analytical studies were conducted in [18] and [19] demonstrating the superiority of Hierarchical MECs (H-ECs)' over typical flat MECs in terms of delay and task admissibility. However, installation of multiple proprietary cloudlets and their organization in H-MECs incur considerable capital and operational expenditures (*e.g.* [20]). These challenges motivate the necessity of an alternative cost-minimal solution that accounts for both latency requirements and the tasks' admissibility. This level is where Vehicular Fog Computing (VeFC) and the concept of VaaR come into play.

Over a long time, if the vehicles dedicate their OBUs only for their internal processing, their capabilities will probably be underutilized. As these vehicles' OBUs possess AI tools, other vehicles that lack such resources can improve their driving automation potentials through wireless computational tasks offloading. One of the typical examples in driving automation is Augmented Reality (AR), where the system creates 3D objects to support the driver's safety and avoid traffic jams. A vehicle can request from other vehicles to sense their surrounding environments, analyze, and send back the results to the requester to support the creation of accurate AR objects that support both short-term and long-term navigation. A vehicle that tries to choose between lanes can request from other vehicles to evaluate their lanes condition based on received or collected information to support the decision of the requesting vehicle. Besides, IoT) devices and pedestrians may also benefit from the computational capabilities available on OBUs. For example, IoT devices that require a road network status may provide vehicles with data or ask vehicles to execute some artificial

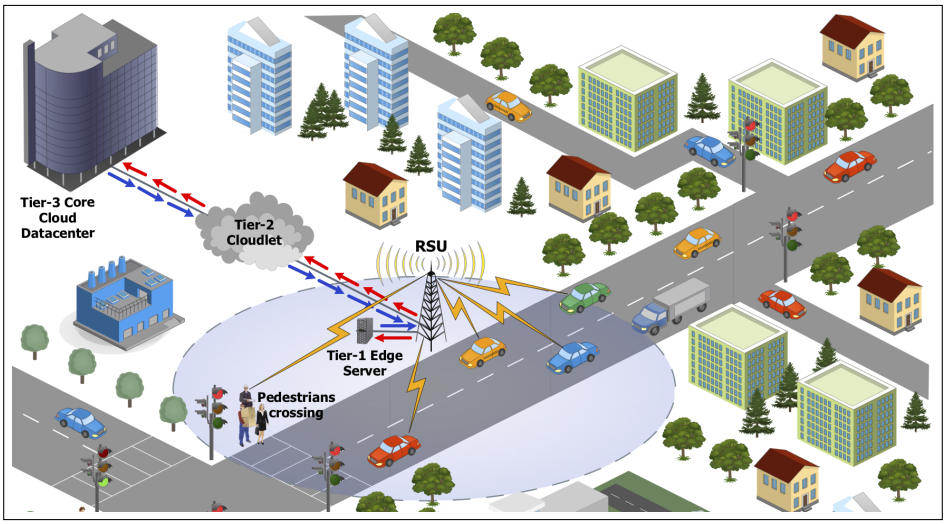


Figure 4.1: Hierarchical MEC-based sub-networking scenario.

intelligent tasks over some data and send results back to them. A pedestrian who tries to apply a heavy task, such as object recognition over a picture, can get help from nearby vehicles to perform such an energy-intensive and complex task instead of doing it locally.

4.2 System Model

We consider a network scenario as depicted in Fig. 4.1; an RSU is located in a dense urban area and provided with wireless communication capabilities allowing it to communicate with vehicles present in its communication range.

The RSU is assumed to be equipped with edge computing capabilities and renders services to incoming requests. Requests arrive at the RSU and ask for computational processing and are assumed to be emanating from either incoming vehicles, requesting particular computing and processing beyond the resources available on the vehicle, or pedestrians or IoT devices whose computing capabilities are limited. The RSU schedules tasks and assigns them resources over the cloudlet co-located with the RSU or over in-range vehicles (servers) with available resources demanded by the tasks.

Here, it is assumed that some of the in-range vehicles may have computing capabilities that we can leverage to offload the tasks awaiting processing. Therefore, our objective is to schedule the processing of incoming tasks' requests either at the RSU or at in-range vehicles and assign them enough computing resources to complete processing within their deadlines.

4.2.1 Communication Model

The network operates on a radio spectrum allocated for the communication between the clients and servers; the total spectrum width is assumed to be B, and both uplink and downlink transmissions are assigned portions of this bandwidth. Transmissions are assumed to be, for simplicity, orthogonal to avoid interference. The downlink and the uplink bandwidth portions are α and β respectively ($\alpha \leq B$ and $\beta \leq B$). In our model, we decide the portion of bandwidth allocated to each of the links. Let $r(t)$ be the rate achieved on the link between a server and a client at time t. Then, $r(t)$ is a function of the radio spectrum allocated to the link and the distance $d(t)$ separating the client from the server at time t.

$$r(t) = \alpha_t \times log_2(1 + \frac{P \times d(t)^{-\sigma}}{I_r + N_0 * \alpha_t}) \tag{4.1}$$

where P is the transmission power, N_0 is two times the power spectral density, $d(t)$ is the distance between the source and destination at time t, σ is the path loss exponent, and I_r is the interference.

Now suppose the client uploads (download) a task of size u to/from the server, the time it takes to offload this task is $T_u = \frac{u}{r}$ where r is the transmission rate achieved during the offload, which is a function of the instantaneous rate on the link. Hence, the equation that relates the upload time T_u, the task size u and the rate $r(t)$ is:

$$u = \int_{t_0}^{T_u + t_0} r(t)dt \tag{4.2}$$

t_0 is the time a client starts offloading the task. $r(t)$ depends on the distance $d(t)$ between the server and the client, where $d(t) = \left[(t \times v_c^x - t \times v_s^x)^2 + (t \times v_c^y - t \times v_s^y)^2\right]^{0.5}$. Here, $v_c^x = v_c\cos(\theta_c)$ ($v_c^y = v_c\sin(\theta_c)$) and $v_s^x = v_s\cos(\theta_s)$ ($v_s^y = v_s\sin(\theta_s)$) are the x-component (y-component) of the client and the server velocities, v_c and v_s, respectively and θ_c (θ_s) is the angle between $v_c(v_s)$ and the x-axis. t is the time that has elapsed from the start of the upload time and (0,0) being the entry point to the coverage of the RSU. The complex integration above is difficult to solve and particularly for this dynamic environment and online operation. To overcome this difficulty, we calculate the distance between the clients and the servers in the middle of each time slot and compute the rate accordingly.

4.2.2 Computation Model

Our system assumes a server is available at the RSU and each vehicle $j \in J$. The computational capacity of each vehicle is depicted by its CPU frequency f_j. For simplicity, we assume that each vehicle's OBU server contains only one CPU. Each task $i \in I$ is characterized by five deterministic values: upload and download data sizes γ_i and σ_i respectively with their computation requirement c_i, weight ω_i which represents the importance of the task to be scheduled before its deadline, and deadline δ_i. The tasks are indivisible; they can not be partitioned and should be computed by only one server. A server can not start adding a task unless it receives all the required data and can not send back the calculated result unless it completes the task computation.

4.2.3 Problem Definition

The problem input is a set of tasks $I = \{1, ..., i, ..., n_i\}$ required to be offloaded from several sources to a set of servers J available at the RSU and the vehicles' OBUs $S = \{0, 1, ..., j, ..., n_j\}$. Task i requires a certain application available only on a subset of these servers, say J_i, and it is characterized by its computational cycles required c_i, its upload data size γ_i, its download data size σ_i, its deadline δ_i and its weight ω_i. The aim is to find the amount of computing resources assigned from server j to task i $f_{t,ij}$ in each time unit t. Also the radio resource/bandwidth $\alpha_{t,ij}$ (respectively $\beta_{t,ij}$) dedicated for the link used for transmission of task i client and server j. The objective is to maximize the weighted number of processed offloaded tasks. Note that the processing of a task can not start until all the required data is uploaded to the assigned vehicle. Similarly, a task cannot be downloaded until the processing of the task is finished. We indicate the distance between vehicle j and the source of task i at time unit t by d_{ij}^t.

Proposition 4.1. *The scheduling problem defined above is a NP-hard problem.*

Proof. Consider the well-known NP-hard scheduling problem $P|pmtn|\sum w_i U_i$ [77], where there are N tasks with deadlines to be scheduled over M parallel identical machines to maximize the number of admitted tasks. It is easy to show that instances of this problem can be reduced to our problem. For each task in this problem, we may create a task for our problem having the same specifications without being implicitly downloaded or uploaded to a certain vehicle. Then we create M machines similar to the above known NP-hard problem to process the maximum number of admitted tasks during the entire timeline. We reduced the scheduling problem $P|pmtn|\sum w_i U_i$ to our problem in polynomial time; hence our problem is an NP-Hard problem. □

We, therefore, mathematically formulate the problem as a mixed-integer linear

program. Then we propose a decomposition scheme based on the Dantzing-Wolfe decomposition method in Section 6. This method divides the problem into a linear master problem and multiple pricing sub-problems, which can be solved in polynomial time through dynamic programming.

4.3 Mathematical Model

In this section, we formulate the problem as an ILP. [2]

Table 4.1: Symbols used in formulating the problem

Symbol	Explanation
a) Parameters	
I	Set of offloaded tasks
J	Set of servers
J_i	Set of servers that have the application of task i
T	Maximum time segment
B	Total spectrum
c_i	Required number of clock cycles for task i
δ_i	Deadline of task i
γ_i	Upload data size of task i
σ_i	Download data size of task i
d_{ij}^t	Distance between vehicle j and the RSU
Δ	Time unit length
ω_i	Weight of task i
f_j	Maximum computation capacity of server j
N_0	Power spectral density
α	Path loss exponent
b) Variables	
$f_{ij} \in \mathbb{R}$	Frequency assigned for task i on server j.
$us_i^t \in \{0,1\}$	Task i is in upload stage at time t or not.
$ps_i^t \in \{0,1\}$	Task i is in processing stage at time t or not.
$ds_i^t \in \{0,1\}$	Task i is in download stage at time t or not.
$\alpha_{t,ij} \in \mathbb{R}$	Bandwidth assigned for i to be uploaded to server j.
$\beta_{t,ij} \in \mathbb{R}$	Bandwidth assigned for i to be downloaded from server j.
$x_{ij} \in \{0,1\}$	Indicates whether task i is assigned to server j or not.
$rd_{t,ij} \in \mathbb{R}$	Download data rate of task i from server j at time t.
$ru_{t,ij} \in \mathbb{R}$	Upload data rate of task i to server j at time t

[2]The list of symbols is shown in Table 4.1

The objective of the mathematical model is:

$$maximize \sum_{i \in I} \sum_{j \in J_i} \omega_i x_{ij} \tag{Obj1}$$

In words, maximize the weighted number of tasks scheduled over the servers, subject to the following constraints:

1) Sum of computation resources assigned to the offloaded tasks to a server can not exceed maximum capacity.

$$\sum_{i \in I} f_{t,ij} \leq f_j \qquad \forall j \in J \qquad \forall t \leq T \tag{C1}$$

2) The computation resources assigned to a task from one server should be sufficient to finish the task.

$$\sum_{t \leq T} f_{t,ij} \times \Delta = x_{ij} c_i \qquad \forall i \in I \qquad \forall j \in J_i \tag{C2}$$

3) A task, if scheduled, is either in upload, compute, or download stage.

$$us_i^t + ps_i^t + ds_i^t = \sum_{j \in J} x_{ij} \qquad \forall i \in I \qquad \forall t < \delta_i \tag{C3}$$

4) The first time unit in the lifetime of each task must be in the upload stage unless it is decided to be rejected.

$$us_i^1 = \sum_{j \in J} x_{ij} \qquad \forall i \in I \tag{C4}$$

5) The last time unit before the deadline of a task must be in the download stage

unless it is decided to be rejected.

$$ds_i^{\delta_i} = \sum_{j \in J} x_{ij} \qquad \forall i \in I \tag{C5}$$

6) Throughout the time, a task can not be in an upload stage unless it was in the upload stage in the previous time unit.

$$us_i^t \leq us_i^{t-1} \qquad \forall i \in I \qquad : 2 \leq t \leq \delta_i \tag{C6}$$

7) In any time unit, a task can not be in a download stage unless it will be in the download stage in the next time unit (unless it is rejected).

$$ds_i^t \leq ds_i^{t+1} \qquad \forall i \in I \qquad \forall t < \delta_i \tag{C7}$$

8) After the deadline, no activities should be in progress.

$$us_i^t + ps_i^t + ds_i^t = 0 \qquad \forall i \in I \qquad \forall t \geq \delta_i \tag{C8}$$

The bandwidth used for all the transmissions happening in one time unit can not exceed the total spectrum

$$\sum_{\substack{i \in I \\ j \in J}} \beta_{t,ij} + \sum_{\substack{i \in I \\ j \in J}} \alpha_{t,ij} \leq B \qquad \forall t \leq T \tag{C9}$$

9) A task can not be assigned a computation resource from a vehicle unless it is in the computation stage.

$$f_{t,ij} \leq f_j \times ps_i^t \qquad \forall i \in I, \forall j \in J_i, \forall t \leq T \tag{C10}$$

10) In each time unit, a task can not be assigned a bandwidth to download data unless it is in the download stage.

$$\beta_{t,ij} \leq B \times ds_i^t \qquad \forall i \in I, \forall j \in J_i, \forall t \leq T \tag{C11}$$

11) In each time unit, a task can not be assigned a bandwidth to upload data unless it is in the upload stage.

$$\alpha_{t,ij} \leq B \times us_i^t \qquad \forall i \in I, \forall j \in J_i, \forall t \leq T \tag{C12}$$

12) The download transmission rate for task i is :

$$r_{ij}^{dt} = \beta_{t,ij} \times log\left[1 + \frac{p_i \times (d_{ij}^t)^{-\sigma}}{\beta_{t,ij} \times N_0}\right] \tag{4.3}$$

The upload transmission rate r_{ij}^{ut} can be calculated similarly.

13) For each task, the bandwidth should be assigned for the entire data to be uploaded or downloaded.

$$\sum_{t \leq T} r_{ij}^{ut} \times \Delta = \gamma_i \times x_{ij} \qquad \forall i \in I \qquad \forall j \in J_i \tag{C13a}$$

$$\sum_{t \leq T} r_{ij}^{dt} \times \Delta = \sigma_i \times x_{ij} \qquad \forall i \in I \qquad \forall j \in J_i \tag{C13b}$$

Note that constraints C13a and C13b are non-linear. To linearize them, we apply a well-known technique that is based on the calculation of the gradient of the bit rate function.

Finally, a task should only be assigned to one server.

$$\sum_{j \in J} x_{ij} \leq 1 \qquad \forall i \in I \tag{C14}$$

4.4 Dantzig-Wolfe Decomposition

Dantzig-Wolfe decomposition technique utilizes an ILP property that several constraints of a problem contain only a subset of the variables (Blocks structure). These constraints, if separated, define an easy-to-solve subproblem. The approach uses the presentation theorem of linear programming to encode all the feasible points of the original problem as an affine combination of the subproblem feasible points.

Now, looking into our problem model, we can infer that only two constraints have variables of different tasks ($C1$ and $C9$). All the other constraints contain variables of one task. Hence, using Dantzig-Wolfe decomposition, we can decompose the problem into one master problem and N pricing subproblems.

Let X^i be the set of points that satisfies all the constraints of task i except constraints in the sets $C1$ and $C9$. Then any feasible solution for the problem can be written as affine combinations of the points in x_{ij} under the condition that it satisfies $C1$ and $C9$. Let K_i be the set of integer points in the task i feasible space. Let $\mathbf{x}_i^k$ be feasible integer point in task i feasible space. Then the problem can be re-written as follows:

$$maximize \sum_{i \in I} \sum_{j \in J_i} \omega_i \sum_{k \in K_i} \lambda_i^k x_{ij}^k$$

$s.t.$

$$\sum_{i \in I} \sum_{k \in K_i} \lambda_i^k f_{t,ij}^k \leq f_j \qquad \forall j \in J \quad \forall t \leq T \tag{C1'}$$

$$\sum_{\substack{i \in I \\ j \in J}} \sum_{k \in K_i} \lambda_i^k \left(\beta_{t,ij}^k + \alpha_{t,ij}^k\right) \leq B \qquad \forall t \leq T \tag{C9'}$$

$$\sum_{k \in K_i} \lambda_i^k = 1 \qquad \forall i \in I \tag{Ca}$$

$$\lambda_i^k \in \{0, 1\}$$

$$\mathbf{x}_i^k \in X_i$$

As there is an exponential number of combinations of points in each X_i, an efficient way to solve the problem is by relaxing the variables to linear ones and solving the problem via column generation (CG) [78]. The following subsection will discuss our CG approach.

4.4.1 The Column Generation Algorithm

As discussed in the previous section, the possible number of columns combinations of the master problem is exponential. The basic idea of the column generation algorithm is to avoid including all the possible columns of a problem in the master model tableau. This avoidance is done by making the optimization of the master program to calculate the dual variables and feed them to the pricing subproblems. The role of the subproblems is to generate the columns with the minimum reduced cost using the given dual values. When the master model converges according to a certain criterion, the algorithm starts to solve the integer version of the master problem.

4.4.1.1 Initial solution

We chose to provide an initial solution with a constructive greedy heuristic. The algorithm pseudocode is shown in Algorithm 5. The algorithm starts by sorting the tasks according to their processing times in ascending order. Then for each task, it sorts the vehicles' OBUs according to their available computation resources. Then for each server, it tries to assign that task. If it succeeds, it considers the task is scheduled. Otherwise, it resets the resources as the task was not scheduled. The input of the greedy algorithm is the set of tasks $\boldsymbol{I}$, the set of servers $\boldsymbol{J}$ and the transmission rate between each task-server pair in each time unit $\boldsymbol{R}$.

Proposition 4.2. *The time complexity of the greedy approach in Algorithm 5 is* $O(N\big(M\log(M)+MT\big))$.

Proof. We can sort the tasks with merge sort with complexity $O\big(N\log(N)\big)$. Since for each task, we are sorting the servers, this will take $O(NM\log(M))$. Checking for available server whether the task can fit or not will take $O(NMT)$. So the total complexity will be $O(O\big(N\log(N)\big)+N\big(M\log(M)+MT\big))$ and since the second term is more complex, we can neglect the first term. □

Proposition 4.3. *The greedy approach in Algorithm 5 always returns a feasible solution.*

Proof. We prove this with loop invariant. The initial step is when the algorithm tries to schedule the first task. Since the resources are vacant, the algorithm will not find a problem in detecting the available resources. Hence, all the "if" conditions on lines 10, 16, and 23 will be satisfied. If the task requirement is big enough that all the available resources can not satisfy it, the function $DeleteTask$ will free all the resources acquired by this task. Otherwise, the task will be admitted. Up to this point, the solution is feasible. Now, for all the other tasks, the "if" conditions on line

10, 16, and 23 will check whether the resources in a certain time unit is available or not. If available, the new task will be assigned these available resources; otherwise, it will check other time units. This will avoid assigning two or more tasks the same resource at the same time. Again, if the task requirement is completely satisfied, the task is admitted. Otherwise, the function $DeleteTask$ will free all the resources the current task has acquired. After going through all tasks, the algorithm will terminate having a certain number of tasks being admitted and another rejected without any overlap between the assigned resources. Hence the final solution is feasible. □

4.4.1.2 Solving the Subproblems

Let ψ_{jt} be the dual variable corresponding to a constraint in the set $C1$, ϕ_t be the dual variable corresponding to constraint in set $C9$, and ζ_i be the dual variable of constraint Ca for task i. Then the column generation pricing subproblem for each task is modeled as:

Algorithm 5 Greedy algorithm

```
1: procedure GREEDY(I, J, R)
2:     x_ij ← 0  ∀i ∈ 𝓘
3:     f̂_j ← T × f_j  ∀j ∈ J
4:     sort(I, c_i ≤ c_{i+1})
5:     bandwidthAcquired[1...T] ← false
6:     compAcquired[1...|J|][1...T] ← false
7:     for i ∈ I do
8:         uploaded ← 0
9:         downloaded ← 0
10:        computed ← 0
11:        sort(J, f̂_j ≥ f̂_{j+1})
12:        for j ∈ J do
13:            for t ← 0 : δ_i do
14:                if uploaded < γ_i then
15:                    if !bandwidthAcquired[t] then
16:                        uploaded += R[i][j][t] × Δ
17:                        bandwidthAcquired[t] ← true
18:                        β_{t,i,j} ← B
19:                else if computed < c_i then
20:                    if !compAcquired[j][t] then
21:                        computed += f_j × Δ
22:                        compAcquired[j][t] ← true
23:                        f_{t,ij} ← f_j
24:                else if downloaded < σ_i then
25:                    if !bandwidthAquired[t] then
26:                        downloaded += R[i][j][t] × Δ
27:                        bandwidthAcquired[t] ← true
28:                        α_{t,i,j} ← B
29:            if downloaded >= σ_i then
30:                x_ij ← 1
31:            else
32:                DeleteTask(i)
       return {x, f, α, β}
```

$$
\begin{aligned}
minimize & \sum_{\substack{j \in J \\ t \leq T}} \psi_{jt} f_{t,ij} \\
& + \sum_{t \leq T} \phi_t \sum_{j \in J} \left(\alpha_{t,ij} + \beta_{t,ij} \right) \\
& - \omega_i x_{ij} \\
& + \zeta_i
\end{aligned}
\tag{83}
$$

By looking into the subproblem objective function and constraints, we can state the subproblem as follows:

Definition: Given a set of time units, each with its own weights for being used only to upload, download or compute the task in every server, specify the process (uploading, downloading, or computing) and the amount of resource used in each time unit in one server to minimize the total weight.

The solution space of this problem is polynomial in size. Hence, it is possible to solve the problem using a dynamic programming approach that can efficiently find the optimum solution without the need for any branch-and-bound-based algorithm.

Function 1 Get feasible solution for one stage

```
1: function FEASIBLE(A, amount, price, il, fl, inc)
2:     stage.index ← il
3:     while stage.amount < A AND
4:     stage.index <= fl do
5:         stage.addCurrentIndex(price[stage.index] / amount[stage.index])
6:         inc(stage.index)
7:     removeExtraPrice(stage)
8:     return stage
```

Function 2 Remove the extra time units.

```
1: function UPDATESTAGE(stage, amount, A)
2:     index ← stage.index()
3:     maxAmount ← amount[index]
4:     while stage.amount − maxAmount > A do
5:         stage.deleteItem(index)
6:         index ← stage.index()
7:         maxAmount ← amount[index]
8:     removeExtraPrice(stage)
9:     return stage
```

Our solution to the subproblem of a specific server-client pair is shown in algorithm 6. There are three loops in the algorithm (lines 5, 9, and 14). Before the outer loop, the algorithm calls *feasible* (shown in Function 1). This function starts from a time

Algorithm 6 Subproblem Solution

```
1: procedure SolveSubproblem(i, j, ψ, φ, f_j, R)
2:
3:     s.up ←
4:         feasible(γ_i, R[i][j] * Δ, φ * S, 0, δ_i, ++)
5:     while s.up.index < δ_i do
6:
7:         s.down ← feasible(σ_i, R[i][j] * Δ,
8:                 φ * S, δ_i, s.up.index − −)
9:         while s.down.index
10:        > s.up.index do
11:
12:            s.proc ← feasible(c_i, f_j * Δ,
13:                ψ_j * f_j, s.up.index, s.down.index + +)
14:            while s.proc.index < s.down.index do
15:
16:                if s.price > bests.price then
17:                    bests ← s
18:
19:                maxRatio ← s.proc.maxRatio
20:                ratio ← ψ[s.procIndex] / Δ
21:                if maxRatio > ratio then
22:                    s.proc.addCurrentIndex(ratio)
23:                    updateStage(s.proc, f_j, c_i)
24:                s.proc.index + +
25:
26:            maxRatio ← s.down.maxRatio()
27:            ratio ← φ[s.down.index]*S / (rate[i][j][s.down.index]*Δ)
28:            if maxRatio > ratio then
29:                s.down.addCurrentIndex(ratio)
30:                updateStage(s.down, rates[i][j], σ_i)
31:            s.down.index − −
32:
33:        maxRatio ← s.up.maxRatio()
34:        ratio ← φ[s.up.index]*S / (rate[i][j][s.up.index]*Δ)
35:        if maxRatio > ratio then
36:            s.up.addCurrentIndex(ratio)
37:            updateStage(s.up, rates[i][j], γ_i)
38:        s.up.index + +
```

unit given by the parameter *il* and adds the resources available in this time unit into a certain stage (the variable *stage*). While iterating with a certain direction (the *inc* operator) of the timeline, the function keeps adding resources to the stage until it fulfills the task's stage requirement (the variable A) or it passes the possible time unit that can be acquired (fl). For each time unit added, it calculates its price-amount ratio and adds it to a sorted data structure (i.e., a red-black tree) for a purpose explained later. Once the function is done from adding all the necessary time units, the function calls another function called *removeExtraPrice*. This function gets from the stage the time unit with the highest price-amount ratio and keeps only the necessary amount of its resource, and removes the rest from the stage. The first loop in Algorithm 6 iterates starting from the last time unit in the upload stage to the end of the timeline. For these time units, the algorithm calls the function *(*feasible) given the index of the upload stage as the initial time unit. After finding a feasible solution for the download stage, the algorithm starts another loop iterating over all the time units available for downloading. In this loop, the algorithm finds a feasible solution for the processing stage. The algorithm's inner loop iterates from the time unit in the processing stage feasible solution until the index of the download stage. For each time unit, it calculates its price-amount ratio and checks whether it is smaller than the maximum price-amount ratio in the processing stage data structure. If it is the case, it adds the time unit to the data structure and calls *updateStage* shown in Function 2. This function removes all the extra time units having the highest price-amount ratio until it reaches the exact requirement of the stage. Once done from the third loop, the algorithm updates the same way the download stage and decrements its index. Once done from all the time units available for the download stage, the algorithm updates the upload stage by applying the same method applied for the download stage and the processing stage and then increments its index and repeats the entire process.

Proposition 4.4. *The time complexity of the subproblem algorithms is* $O(T^4)$

Proof. We start the proof by analyzing the third loop (lines 14-32). In the worst-case scenario, comparing the current solution with the best solution will always lead to copying the current solution (lines 20-23), leading to the complexity of $O(T)$. Adding a new element to the data structure can be implemented with complexity $O(T)$ (line 29). The function *updateStage* upper bound is $O(T)$ as well (line 30). So for one iteration, the time complexity is $O(T)$ and since in the worst case of the number of iterations is T. Then the complexity of the loop is $O(T^2)$. Now, the second loop (lines 9-33) contain the third loop ($O(T^2)$), calculates the feasible solution of the processing stage (line 12), which can be done in $O(T)$ and updates the download stage time units (lines 26-31) which takes $O(T^2)$ leading to total complexity of $O(T^3)$. With a similar analysis to the first loop (line 5), we can conclude that the total complexity is $O(T^4)$. □

Proposition 4.5. *The subproblem algorithm always return the optimal solution.*

Proof. Assume that the length of each stage is given. Then, each stage has a set of time units that it should choose from to fulfill its requirement while minimizing the cost given by the dual values. This problem is called the *fractional minimum cost knapsack* problem and is solvable in polynomial time. To choose the time units, we calculate its cost-effectiveness ratio (price-amount), then sort the time units and select the ones with the smallest ratios. The last time unit chosen should be partitioned and takes only the required amount of it, so the required resources do not get exceeded hence increasing the cost.

The algorithm starts by finding the shortest time that can be assigned to the upload stage to fulfill its requirement (calling the *feasible* function). After that, it checks all the possible combinations for the other two stages by going through all the

time units available for the download stage and accordingly checking all the available time units for the processing stage. After adding a new time unit for any stage, the algorithm applies the following recursive equation:

if $(\frac{newUnit.price}{newUnit.amount}$

$< \frac{currentUnit.price}{currentUnit.amount})$ **then**

$$\begin{aligned} price(t) = & \, price(t-1) \\ & - price(remove(stage, newUnit.amount)) \\ & + newUnit.price \end{aligned}$$

else

$$price(t) = price(t-1)$$

where the function $remove(s, a)$ takes a stage s and removes the time units with maximum price with total and amount equals a. To implement that, the algorithm calculates the cost-effectiveness ratio of the time unit. Suppose one of the time units currently in the solution has a higher ratio. In that case, it adds the new time unit to the solution and removes any unnecessary time units (extra) from the solution. By doing so, the algorithm will always find the set of time units that has the minimum cost-effectiveness ratio for any length of any stage. Since the algorithm checks all possible lengths of all the stages, then the algorithm will definitely find the optimum solution. □

4.5 Performance Evaluation

We study here the performance of the proposed methods, including Mix Integer Linear Programming (MILP) (to get the optimal solution), Dantzig-Wolfe decomposition method, and greedy algorithm (Greedy). We consider different performance metrics

Table 4.2: Measurable factors used for the scheduling performance

Factors	Distribution	Mean	Variance
Tasks Arrival (tasks/s)	Exponential	4	-
Servers Capacity (GHz)	Gaussian	3	0.2
Vehicle's Velocity	Trunc. Gaussian	25	7
Vehicle's Arrival (m/s)	Geometric	p = 0.1	-
Deadline (s)	Gaussian	0.1	0.03
Upload Data Size (MB)	Gaussian	1	0.25
Download Data Size (MB)	Gaussian	0.1	0.1
Tasks Weight	Gaussian	5	3
Number of Cycles (Millions)	Gaussian	20	5
Total Spectrum (GHz)	-	3	-

such as execution time, performance gap, and weighted rejection rate. We then study the overall weighted rejection rate for the overall system using the Dantzig-Wolfe decomposition method by varying tasks arrival rate, the vehicles emitting probability, the average upload data size, the average number of computational cycles, and the average server size. Vehicular traffic traces are obtained using the well-known traffic simulator SUMO (see Appendix B.1). Table 4.2 shows the parameters used throughout this section, and the probability of application availability for a certain task on a particular server is set to 0.75. We assume an RSU with a $500m$ coverage range, equipped with one server to process offloaded workloads. We use CPLEX (see Appendix **??**) to solve our optimization models and C++ to simulate the operation of our algorithms through a discrete event-driven simulation. We generate results on CPU with Intel(R) Core(TM) i7-6700 CPU @ 2.7GHz, 16GB memory ram, and 64-bit mac operating system. The results are averaged over ten runs.

Table 4.3: Comparison between the algorithms performance in terms of rejection rate and computation time.

Number of Vehicles	Number of Tasks	MILP		Dantzig-Wolfe			Greedy		
		Rej. Rate	Time (s)	Rej. Rate	Dev.	Time (s)	Rej. Rate	Dev.	Time (s)
5	50	0.44%	8	18.70%	19.93%	2	86.70%	86.65%	0.38
	75	2.23%	197	20.35%	18.53%	4	92.31%	92.132%	1
	100	4.34%	66435	23.51%	18.73	47	95.23%	95.21%	4
15	50	0.45%	23.7	19.07%	19.32%	13.25	86.73%	86.66%	1
	75	0.48%	156.85	22.89%	23.24%	29.45	91.11%	91.07%	2.6
	100	0.50%	1378	20.26%	19.95%	70	93.08%	93.04%	6

4.5.1 Scheduling Performance

In this subsection, we analyze and evaluate the performance of our proposed Dantzig-Wolfe decomposition method. First, we study the scheduling performance of our method versus the optimal solution obtained from the MILP by considering the execution time and task rejection rate. We limit the number of iterations for the Dantzig-Wolfe decomposition method to 100 iterations. Table 4.3 shows the execution time of the three methods by varying the problem size as an input instance. Rejection rate = $\frac{\text{weighted num. of admitted tasks}}{\text{total weight}}$, and deviation = $\frac{\text{optimal obj. - weighted num. of admitted tasks}}{\text{optimal obj.}}$. The results are averaged over five samples. From the table, we can observe that, in terms of execution time, even for a small size, the Dantzig-Wolfe decomposition method always surpasses the optimal solution obtained by MILP. For instance, the computational time requires to solve for five vehicles with 100 tasks is over 18 hours which is not computationally acceptable for a real scenario. In contrast, the Dantzig-Wolfe method maintains an acceptable computational time. In addition, the Dantzig-Wolfe method maintains an acceptable deviation from the optimal solution even for large instances (highest deviation is 23% and lowest is 18.5%). Compared to the greedy approach, the Dantzig-Wolfe method can improve the greedy initial solution from 55% to a maximum of 71%. For instance, with 15 vehicles, as shown in the table, the MILP performs relatively much better than the greedy method. This difference is because as the number of vehicles increases, the number of possible solution combinations that will lead to the optimal value increases. The computational time of the Dantzig-Wolfe method gets slower with 15 vehicles compared to 5 vehicles. This is expected since the number of pricing sub-problems required to solve the column generation part of the Dantzig-Wolfe decomposition method is increased. Nevertheless, the execution time compared to the MILP is extremely incomparable. Note that, for all input sizes, the deviation of the Dantzig-Wolfe from the optimal solution

(MILP) is almost the same. Also, notice that we can improve the performance of the Dantzig-Wolfe method by increasing the number of iterations if the system can afford the time given for tasks scheduling.

The study and analysis conducted in this section conclude that the Danzig-Wolfe method is the more suitable and scalable choice for task scheduling in terms of both execution time and task rejection rate. Hence, in the next section, to evaluate the performance of our system through simulation, we use this method for the scheduling stage. But first, it is interesting to show through an example how the Danzig-Wolfe method converges to an optimal solution as proceeding with the number of iterations. Figure 4.2 shows how the feasible solution converges for a 15-task input. The figure shows the upper-bound (i.e., the unfeasible solutions) and the lower-bound (i.e., the best obtained feasible solutions for the master problem after certain iterations) by red thick and black strip lines, respectively. As depicted in the figure, the optimal solution is obtained after 52 iterations which takes one second. The upper bound is obtained by subtracting the reduced costs coefficients of the new columns from the current master objective value. The result is an objective value that can be achieved only if the master accepts all the newly arrived columns; Which is a scenario that can occur only in exceptional conditions.

4.5.2 System Evaluation

In this subsection, we study the performance of our system by simulating the traffic arrival process using the well-known urban traffic simulator SUMO. To get random traces, SUMO requires to specify what is called the vehicle *Emitting Probability (EP)* per second per lane (SUMO is a discrete-time simulator, and this emitting probability is the geometric distribution parameter for each route). In addition, it assumes that the vehicles' speed follows a truncated Gaussian distribution (refer to Table 4.2). The

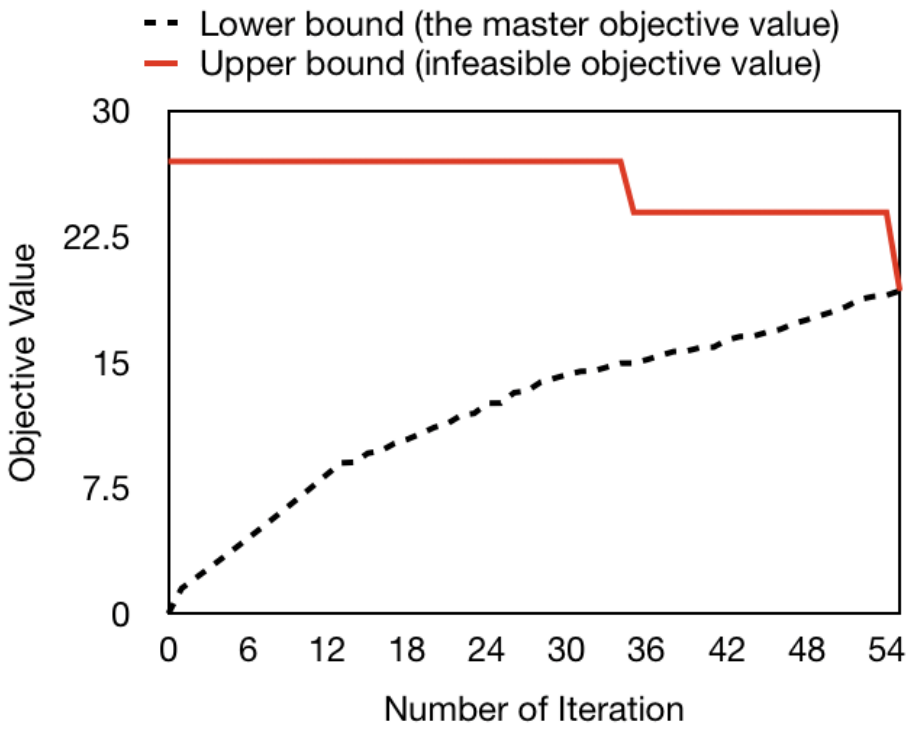


Figure 4.2: The convergence of the columns generation algorithm

event-driven simulation starts by running SUMO for 1 minute (this is the simulation time, not the actual program execution time) to let it reach the steady-state response. Then, we start collecting the information of the available vehicles in the RSU range for one minute. The information collected here contains the vehicles' speed, their coordinate per unit time (which is chosen as one millisecond), their arrival and departure time to and from the RSU range. In addition, we assign each vehicle a certain computation capacity following the distribution specified in Table 4.2. Consequently, we reset the time of the simulator and start generating tasks request events for each vehicle and device following an exponential distribution for the inter-arrival time (refer to table 4.2), and add the events to a sorted data structure (i.e., red-black tree). We generate a scheduling event for each certain period, and eventually, we add it to the data structure. Afterward, the event execution process begins. For each batch of tasks, we start the scheduler, and once we get the scheduling result, we update the

status of the resources. All the results shown in this subsection are averaged over 20 samples.

Fig. 4.3 shows the system performance in terms of weighted rejection rate compared to the different tasks arrival rate per vehicle. Three EP values are considered here (0.1, 0.15 and 0.2). As shown in the figure, from 0.2 to 0.8 task arrival rate, the curves with higher EP values result in a lower rejected rate. Consider here that a vehicle in our system represents a load, as well as a computation resource. Now, when the tasks arrival rate is low, along with the higher arrival vehicles, the overall system capacity maintains a low rejection rate because any vehicle arrival increases the overall system capacity. On the other hand, when the tasks arrival rate is more than one task/s, the system's behavior changes as the arrived vehicles generates load high enough to make the system uses its maximum capacity. Hence, in this situation, the rejection rate increases by increasing the vehicles' arrival rate even when the computational capacity of each vehicle is high.

For a point-to-point comparison, from figure 4.3, we can observe that the plot of Ep=0.2 starts with a lower rejection rate. For instance, with a tasks arrival rate of 0.2 tasks/s, the rejection rates of EP=0.1, EP=0.15, and EP=0.2 are around 8.5%, 9.6%, and 10%, respectively. But, as we increase the tasks arrival rate, the plot of EP=0.2 inflates faster than the others since, with a higher vehicles arrival, more tasks will be generated, and hence the system will be overwhelmed faster. The behavior of the EP=0.15 curve is similar to the EP=0.2 but with a lower steep. From the tasks arrival rate of 1 task/s onwards, the behavior of the two EPs is inverted, and the differences in the number of rejection rates increase with the increase of the tasks arrival rate. For example, at tasks arrival rate 2 tasks/s (or 5 tasks/s), the rejection rate differences between the EP=0.1 and EP=0.15, and between EP=0.15% and EP=0.2% both are 2% (or 4%). As mentioned earlier, the critical point of the system is when the tasks

arrival rate is one tasks/s. Clearly, we can push this point to get higher than one task/s by increasing the system capacity (i.e., increasing the average server size). In practice, this can only be done by increasing the capacity of the RSU server since the system operator has no control over the vehicles computational capacity (either to increase or decrease their capacity).

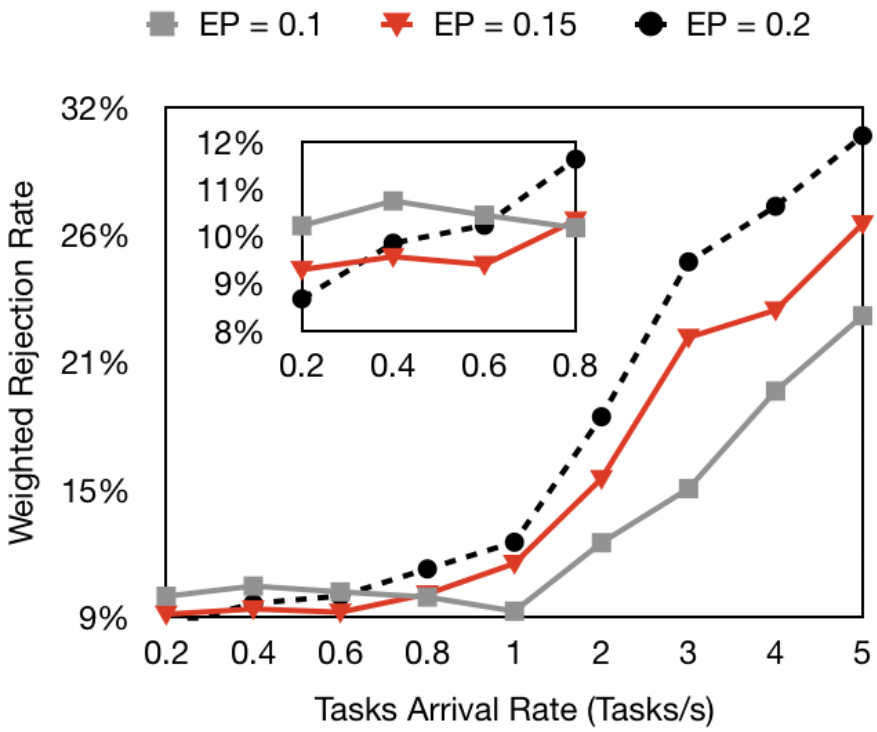


Figure 4.3: Rejection rate versus tasks arrival rate per vehicle.

Figure 4.4 plots the weighted rejection rate of the system versus the emitting probability of the vehicles arrival per lane. The figure shows the system performance for two different tasks arrival rates per vehicles (i.e., 2 tasks/s and 4 tasks/s). While both rejection rates increase at higher EP, the differences in weighted rejection rate between the two tasks arrival rates increase as well. For instance, the relative increase in EP between 0.15 and 0.2 for the tasks arrival rate of 2 tasks/s is 22%, while for the tasks arrival rate of 4 tasks/s is 25%. Consequently, the weighted rejection rate at higher vehicles EP increases linearly. For example, at EP=0.25, the weighted

rejection rate for the tasks arrival rate of 2 tasks/s is 14%, while the tasks arrival rate of 4 tasks/s is 40%. This is due to the fact that the 4 tasks/s arrival rate is reaching the system full-potential usage with higher emitting probability.

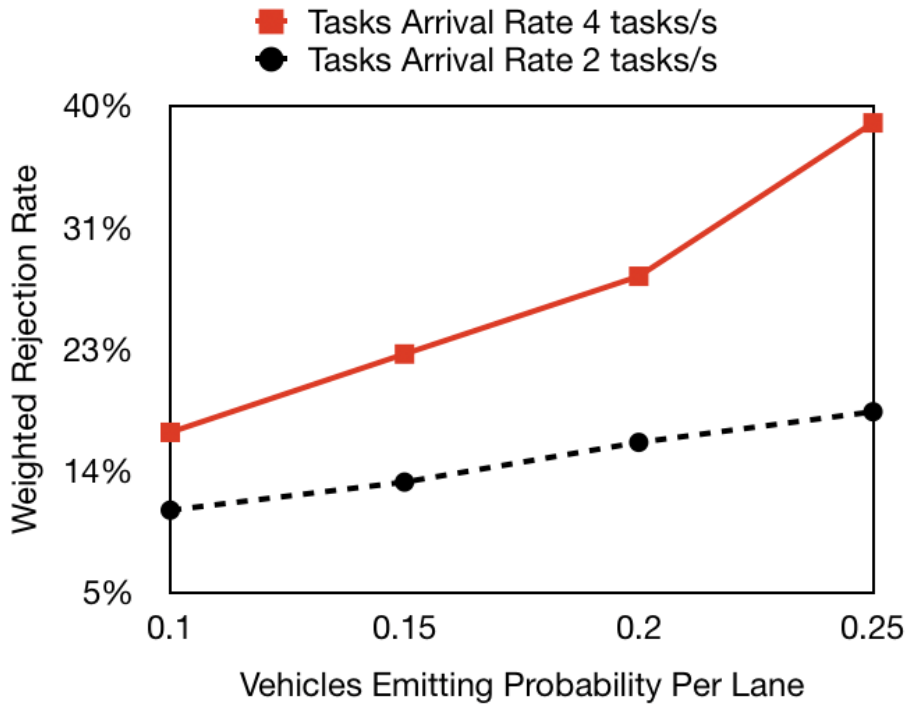


Figure 4.4: Rejection rate versus vehicles arrival rate.

Note that the arrival load of the system can be increased either by increasing the arrival rate or by intensifying the tasks requirements. Figure 4.5 demonstrates the system response in terms of weighted rejection rate as we vary the average upload data size. For example, when the task data size is 1.5 MB, the weighted rejection rate of the system is around 24%, whereas, when the data size of the task is 2 MB, the weighted rejection rate is 26%. This proves that by increasing the data size of arrival tasks upto the wireless bandwidth capacity, the percentage of the weighted number of rejected tasks increases in a nonlinear manner. For instance, the relative increase in weighted rejection rate from 1.5 MB tasks data size to 2 MB is around 6%, while the increase is 10% from 2 MB to 2.5 MB data size.

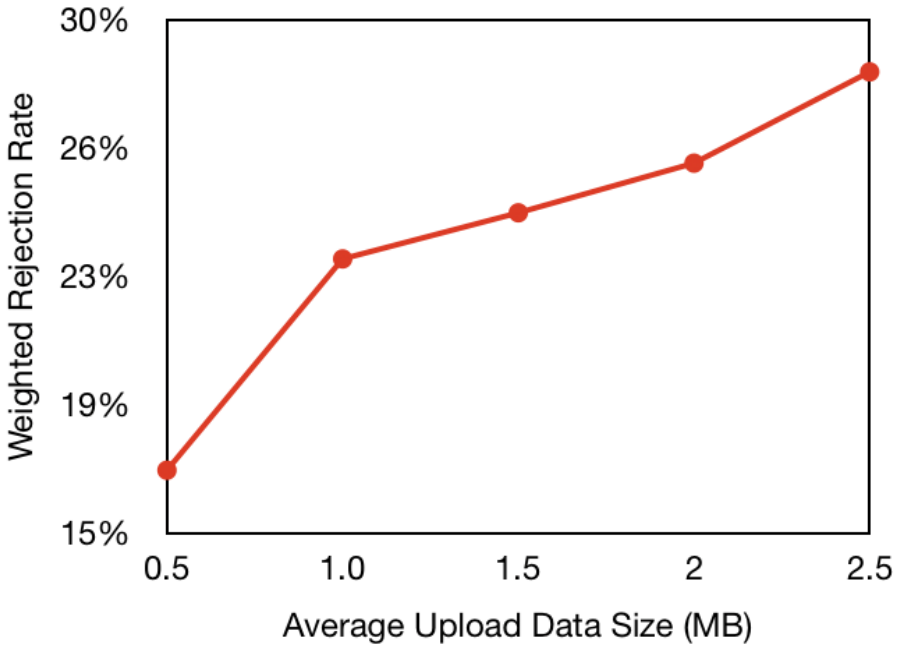


Figure 4.5: Rejection rate versus average upload data size per task.

In Figure 4.6, we calculate the rejection rate for two different average server sizes (2.5 GHz and 3.0 GHz). As depicted in the figure, for a small average number of cycles, the system is capable of admitting almost all tasks, hence, the performance of the two server sizes are almost the same. As the number of cycles per task increases, the difference between the two server sizes starts to increase. For example, for 20 million cycles, the difference between the two server sizes is around 1% while for 30 cycles, the difference reaches more than 2%.

Finally, We study the system performance by increasing the average OBU's server size. Figure 4.7 shows the system performance by increasing the average server size for two different average number of requires cycles (i.e., 10 million cycles and 30 million cycles). As it can be seen from the figure, for smaller server sizes, the difference between the two different number of cycles is relatively large. For example, the difference between the two is around 24% for 1 GHz average server size. By increasing the OBU's server size, the difference between the two different cycles decreases. For

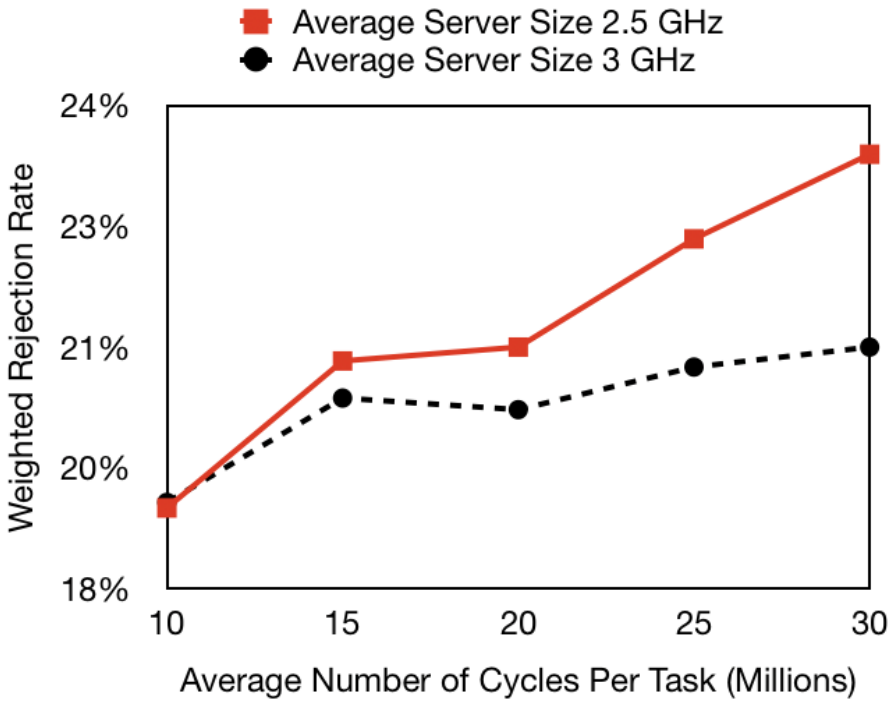


Figure 4.6: Rejection rate versus average number of cycles per task.

instance, with 2 GHz average server size, the difference between the two proposed cycles in system rejection rate is 6%, while for 3 GHz average server size, the difference is 4%. This reduction in the difference of rejection rate between the two different average cycles is due to the fact that both average load sizes can be handled easily when the size of the resource is sufficient enough to handle most of the arriving load.

4.6 Conclusion

Fog computing is a promising paradigm that will allow an efficient utilization of computation resources available and accessible through wireless communication. 5G technologies with its low latency and high reliability promises can provide a platform to enable fog computing establishment upon various kinds of resources including smart vehicles. In this work, we proposed a system that can handle computation

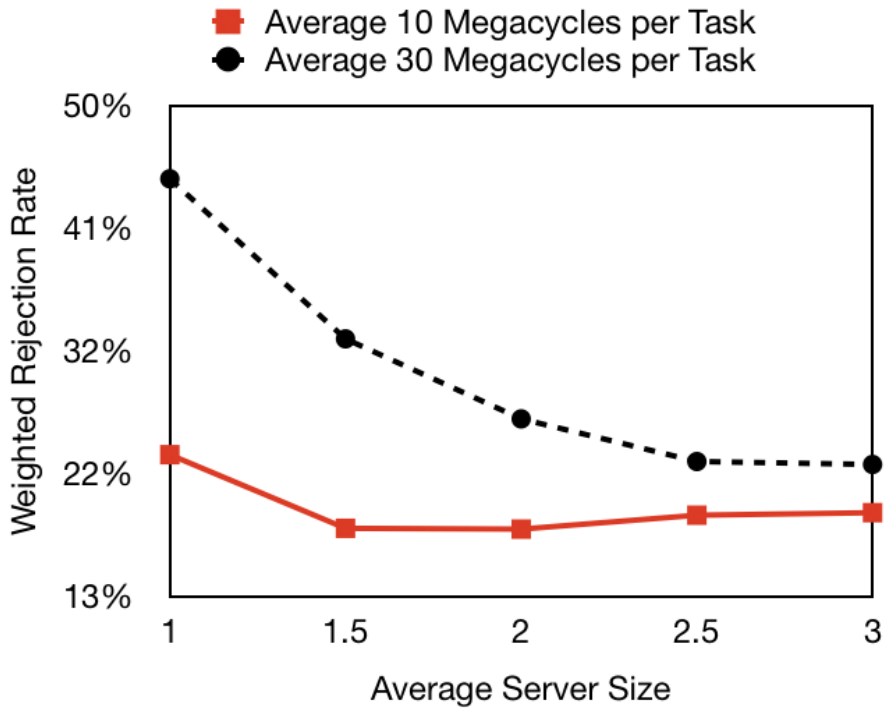


Figure 4.7: Rejection rate versus OBU server average size.

requests over vehicular network through an efficient scheduling scheme that considers the available vehicle OBU computation servers and the limited radio spectrum in order to efficiently allocate them for the requested computational tasks. The problem was formulated as an MILP with the objective to maximize the weighted number of admitted tasks. Although the problem is an NP-hard one, we proposed a scalable decomposition scheme based on Dantzig-Wolfe scheme, which resulted in polynomial-time solvable subproblems and a linear master problem. The approach showed an efficient and scalable performance compared to a greedy heuristic and MILP. Several parameters were considered in the evaluation demonstrating the robustness of our approach to solve various kinds of the problem instances.

Chapter 5

Optimizing Information Freshness for MEC-enabled Cooperative Autonomous Driving[1]

[1]This chapter has been submitted to IEEE Transactions on Intelligent Transportation Systems [79].

As stated earlier, the very objective of this thesis is to study the ability to support/assist the AD environment via a central agent deployed on the edge of the network. The previous two chapters showed that the MEC paradigm does provide abundant computation resources to the vehicles allowing them to offload low-latency computational tasks. This chapter aims to meticulously represent an AD computation workload by portraying the tasks offloaded as relatively long-term processes running in parallel with on-the-road activities. Examples of activities that these processes are supposed to assist are:

1. A vehicle is trying to change its lane. The process will collect information from the vehicle itself and vehicles located on both lanes.

2. Several vehicles are trying to establish a platoon. The process will collect data from all these vehicles and assist them via directives computed continuously.

3. A vehicle is taking a turn.

To be more precise, we consider a network of vehicles, each generating one or more streams of packets to send to processes running at the edge of the network. These processes are assumed to be running applications in the context of a CAD service. The edge server schedules the streams for uplink transmissions and processes to analyze data received. The objective is to achieve this jointly while maintaining the data freshness. Upon completing their processing, the data is broadcast back to the involved vehicles. We formulated the problem mathematically, and it turns out that such a problem is NP-hard and we subsequently applied a decomposition method and heuristics to attain solutions.

5.1 Motivation

Cooperative Autonomous Driving (CAD) has emerged as a platform wherein a vehicle communicates with other vehicles (V2V) or infrastructure (V2I) to cooperatively perform tasks of interest for the AD application. To enable vehicles to cooperate, they leverage available communication technologies (e.g., 5G, and beyond 5G) and communication protocols (WAVE, 5G-V2X, etc.) to perform on-the-road tasks which require detailed information about the status of the surrounding environment. Typical CAD communication relies on establishing a communication channel between vehicles; these vehicles exchange status information (obtained using their sensing capabilities) to help them improve their decision-making. However, this distributed information analytic model suffers from vital issues: 1) A vehicle's OBU might lack the computation capabilities needed to perform certain computational tasks. 2) Establishing a wireless link for each communicating pair wastes a considerable amount of radio resources. 3) An independent decision taken by a vehicle might not suit decisions taken by other vehicles. Therefore, an agent empowered with computing capabilities, gathers the required information, and broadcasts the computed results to all the involved entities, have emerged as an alternative model to tackle these difficulties.

With an MEC server deployed over the RSU, vehicles can offload their workload, as well as other information, sensed from their surrounding, to an agent (running a CAD service) at the edge server. The agent typically runs the service and some learning algorithms to assist the AD application in making decisions on behalf of the concerned vehicles. As indicated earlier, the edge provides abundant resources to cater to services for latency-sensitive ITS applications. While latency has been the metric of utility for most of these applications, this metric lacks the capability of assessing the *freshness* of data, which is of extreme importance to some CAD

applications.

To characterize the information freshness, a new metric has been proposed recently, the *Age of Information* (AoI) metric [80]. In real-time systems (a sensor and a monitor), the AOI measures the time elapsed since the last data packet/measurement was successfully delivered to the monitor. A further interesting extension to the notion of AoI is to consider its *risk level*. Recent works proposed to consider a *threshold* that a system's AoI level should not exceed hence trying to minimize the time of the system being above this threshold [57, 81].

5.2 Contributuins

We summarize the contribution of this work as follows:

- We provide a formal problem definition and prove its NP-hardness. Then, we formulate it as an integer linear program and propose an efficient logic-based Benders decomposition scheme with a polynomial-time solvable sub-problems and an integer master problem.
- We study the method scalability by comparing its performance in terms of outcome and speed with the branch-and-cut algorithm implemented by CPLEX (see Appendix B.2).
- Finally, we examine several system characteristics by varying several system parameters to evaluate the performance of the proposed scheduling method.

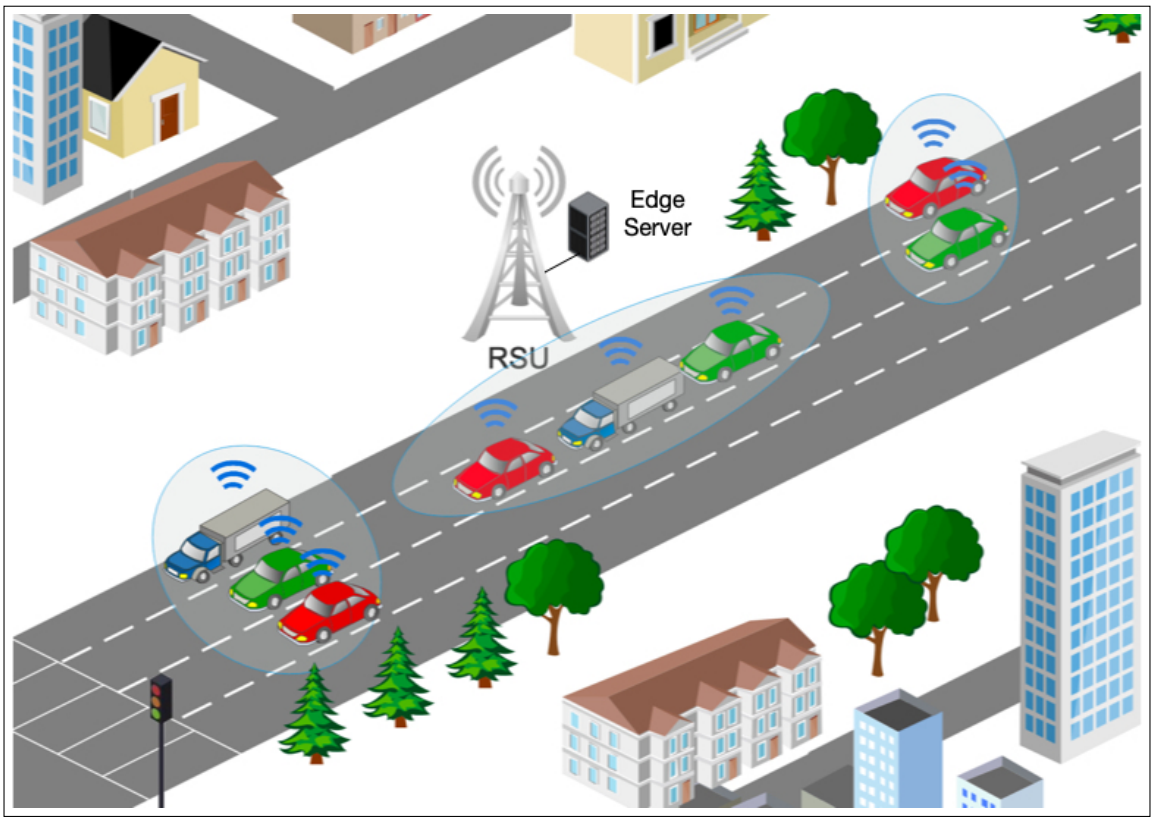


Figure 5.1: A typical scenario of our system model.

5.3 System Model

We consider a scenario as depicted in Figure 5.1; an RSU is located in a dense urban area and provided with wireless communication capabilities allowing it to communicate with vehicles present in its communication range. The RSU is assumed to be equipped with edge computing capabilities (see Figure 5.1). Vehicles are equipped with sensory devices which sample signals and processes in their environment (e.g., speed, direction, surrounding, ...etc.). Each such sensing device generates a stream of data to be offloaded to one or more edge computing processes running a CAD application. The RSU, part of the network, schedules the access of vehicles to the network as well as allocates spectrum and computing resources for their communication and computation. The RSU then relays back the processed data to the vehicles residing in its range. Our objective is to schedule the resources allocation (both radio and computation) over these processes to minimize the threshold-exceeding AoI and thus maintain information freshness.

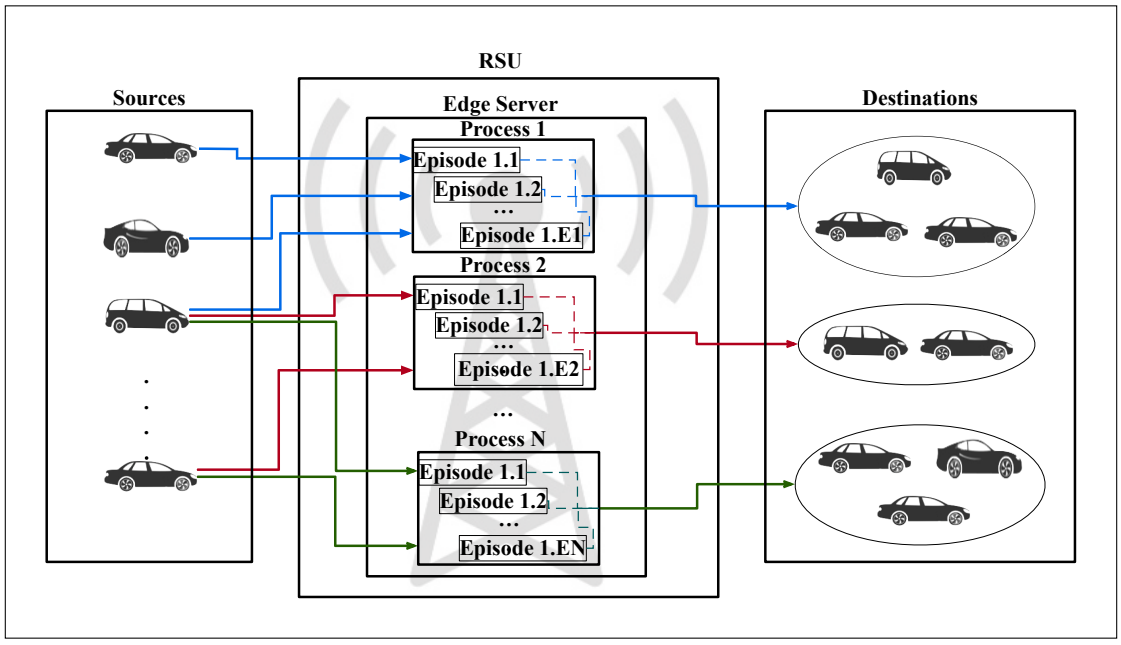


Figure 5.2: Multi-streams multi-processes system.

5.3.1 The Communication Model

The network operates on a radio spectrum allocated for the communication between the vehicles and the RSU; the total spectrum width is assumed to be B, and both uplink and downlink transmissions are assigned portions of this bandwidth. Transmissions are assumed to be, for simplicity, orthogonal to avoid interference, and B is equally divided into several resource blocks $\{r \in \boldsymbol{\mathcal{R}}\}$. The data rate at time t to/from RSU can be calculated as follows:

$$rt(s,t) = \beta * \log_2(1 + \frac{Po * d(s,t)^{-\ell}}{N_0 * \beta}) \,, \tag{5.1}$$

where β is the bandwidth assigned to each resource block, Po is the transmission power, N_0 is the thermal noise, $d(s,t)$ is the distance between source and destination of stream s at time t, ℓ is the path loss exponent.

5.3.2 Computation Model

We assume several virtual machines are available at the edge server $\{m \in \boldsymbol{\mathcal{M}}\}$. The computational capacity of each server is depicted by its CPU frequency f_m. Several processes are running on these virtual machines $\{p \in \boldsymbol{P}\}$. Each of these processes has several data streams $\{s \in \boldsymbol{\mathcal{S}}_p\}$, each has a packet size u_s and a source vehicle v_s. A process p generates one output packet and requires c_p cycles from a virtual machine. Each process has a certain AoI threshold τ_p.

5.3.3 Process-based AoI

The following is the function to calculate the average AoI:

$$\bar{H}(p) = \frac{1}{T_p} \sum_{t \leq T_p} \alpha_{pt} \,, \tag{5.2}$$

α_{pt} is the AoI of process p at time t and T_p is the process duration. The established process aims to support/assist a cooperative autonomous driving application that involves more than one vehicle. As mentioned earlier, the process will receive several data streams from many vehicles and their sensing devices (e.g., camera) (*upload stage*). After that, these streams go through a *computation stage* to obtain useful decisions for involved vehicles. A *download stage* then starts to broadcast the result to the destination vehicles (see Figure 5.2). The metric of interest for assessing the utility of the sampled data is its freshness which is quantified using the AoI metric. The AoI for each stream is determined by the time elapsed since the last successfully received packet of that stream. Normally, the age evolves linearly until the arrival of a new sample, then it is reduced by the same amount spent in the delivery of the packet and starts increasing again. In our CAD application, the calculation of AoI for such a process should count for all the aforementioned stages. Assuming that streams are sampled each time unit, the AoI should be calculated with the stream that started its transmission the earliest.

We define the *episode* of a process as a time interval that includes one distinct sequence of upload, computation, and download stages (see Figure 5.3). A process that calculates the decision/actions that should be sent continuously to the vehicle should go through one or more episodes. Let $\boldsymbol{\mathcal{E}}_p$ be the set of episodes of a process p, $\mathfrak{st}(e)$ is the time unit where episode e starts and $\mathfrak{fi}(e)$ be the time unit where it finishes. Then α_{pt}, AoI of process p at time t, can be calculated as follows:

$$\alpha_{pt} = \begin{cases} \alpha_{p(t-1)} + 1 & \text{if } \forall e \in \boldsymbol{\mathcal{E}}_p \;\; \mathfrak{fi}(e) \neq t\,, \\ t - \mathfrak{st}(e) & \text{if } \mathfrak{fi}(e) = t\,. \end{cases} \tag{5.3}$$

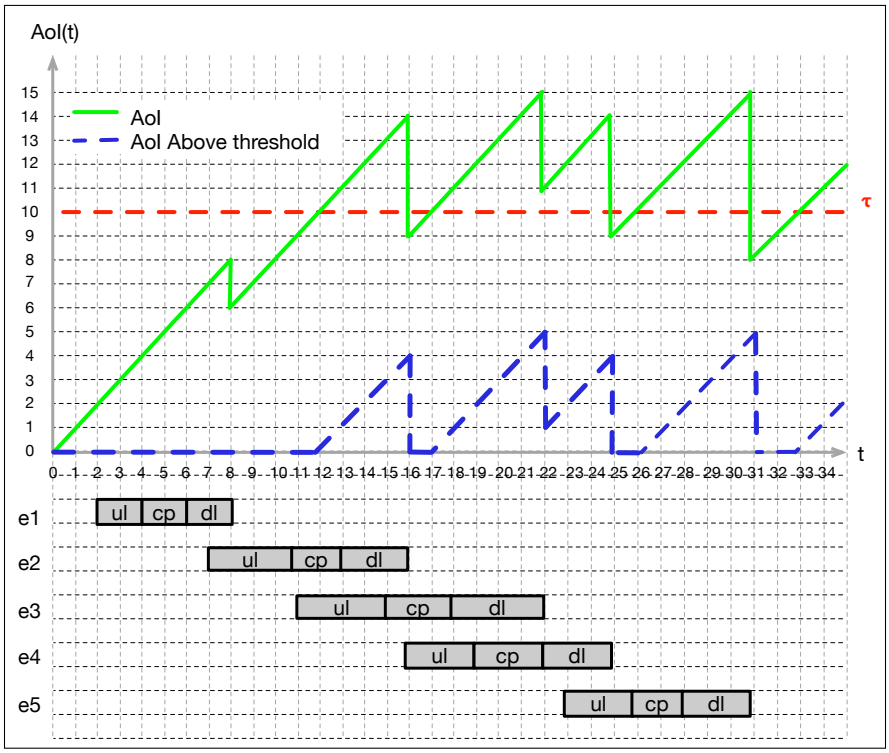


Figure 5.3: The AoI of a process

5.3.4 AoI Risk Level

AD comprises a set of applications and services, each with its own latency and reliability requirements. A pedestrian trying to cross a street by informing vehicles through communication should acquire the highest level of service reliability compared to an application trying to manage the long-term street traffic performance. Hence, it is logical to assume that different processes running at the edge would require a different level of information freshness. Several works already discussed the importance of minimizing the probability of a signal AoI to pass, referred to as the *risk level* [58, 60, 81]. In this work, we assume that each edge process, say p, requires AoI to be below a risk level depicted by a threshold, say τ_p. Thus, we propose to minimize the risk the AoI above threshold, say $\tilde{\alpha}_{pt}$, which is defined as follows: (see Figure 5.3):

$$\tilde{\alpha}_{pt} = \begin{cases} \alpha_{pt} - \tau_p & \text{if } \alpha_{pt} > \tau_p \,, \\ 0 & \text{if } \alpha_{pt} \leq \tau_p \,. \end{cases} \tag{5.4}$$

5.4 Problem Definition and Formulation

We define the scheduling problem formally, prove it is an NP-hard, and mathematically model it as an integer linear program (ILP). This formulation will allow us to propose a scalable solution through logic-based Benders decomposition.

Problem P Definition: *Given a set of processes* $\{p \in \boldsymbol{P}\}$ *all running during a time horizon* T_p*. Each of these processes has several input data streams required* $\{s \in \boldsymbol{\mathcal{S}}_p\}$*, each has a packet size* u_s *and a source vehicle* v_s*. A process* p *requires* c_p *computational cycles from the host virtual machine to generate one outcome. Upon processing, a process sends the outcome to a set of vehicles* $\boldsymbol{\mathcal{V}}_p$*. Each process has a certain threshold of AoI* τ_p *and it is assumed to have a set of episodes to be processed* $\boldsymbol{\mathcal{E}}_p$*. There are several virtual machines* $\{m \in \boldsymbol{\mathcal{M}}\}$ *running on the edge server; each has a certain speed* f_m*. Also, there are available radio resource blocks* $\{r \in \boldsymbol{\mathcal{R}}\}$ *for packets transmitted over the wireless network where each resource block has a bandwidth of size* β*. Find a schedule for the virtual machines and the resource blocks over the processes in each time unit* $t \in T$ *in order to minimize the overall AoI above the threshold of all the processes running.*

Proposition 5.1. *Problem* P *is an NP-hard problem.*

Proof. Consider the well known NP-hard scheduling problem $P|prmtn|\sum T_i$ [82, 83]. This problem objective is to minimize the tardiness of a set of jobs $\{i \in I\}$ by scheduling them over parallel machines $\{m \in M\}$ in a preemptive way. Each job has a processing time c_i and deadline d_i. Any instance of this problem can be reduced to an instance of problem $\mathcal{P}$ as follows: For each each job i, create a process in

our problem with empty input streams set S_p, $d_p = 0$, $c_p = c_i$ and $\tau_p = d_i$ with one one available episode. Solving this instance to optimality will be equivalent to minimizing the time that passed the deadline of the original problem instance, which is the tardiness of the jobs. Since this reduction takes a linear time, then our problem is NP-hard. □

Table 5.1: Symbols used in formulating the problem

Symbol	Explanation
a) Parameters	
$\boldsymbol{\mathcal{V}}$	Set of vehicles
$\boldsymbol{\mathcal{V}}_p$	Destination vehicles of process p
$\boldsymbol{P}$	Set of processes
$\boldsymbol{\mathcal{M}}$	Set of virtual machines
$\boldsymbol{\mathcal{R}}$	Set of resource blocks
$\boldsymbol{\mathcal{S}}_p$	Set of data input streams of process p
$\mp_s$	Set of packets of input stream s
$\mp$	Set of all input streams packets
T	Time frame
B	Total spectrum
u_s	Size of stream s packets
d_p	Packet size of of the output of process p
c_p	Number of cycles required to generate one output packet of process p
T_p	Time horizon for process p
f_m	Speed of virtual machine m
τ_p	AoI threshold of process p
Δ	Time unit length
$\hat{\alpha}_p$	Initial AoI of process p
$v(s)$	Source vehicle of stream s.
$v(p,t)$	The furthest vehicle from the RSU in $V[p]$
b) Variables	
$\alpha_{pt} \in \mathbb{Z}^+$	AoI of process p
$\tilde{\alpha}_{pt} \in \mathbb{Z}^+$	AoI above the threshold of process p
$\rho^m_{et} \in \{0,1\}$	1 if episode e is assigned to machine m at time t and 0 otherwise.
$\omega^r_{\varepsilon t} \in \{0,1\}$	1 if packet ε is assigned to channel r at time t and 0 otherwise.
$\omega^r_{et} \in \{0,1\}$	1 if process p output is assigned to channel r at time t and 0 otherwise.
$s_e[l,t] \in \{0,1\}$	1 if episode e at time t starts stage l where $l \in \{ul, cp, dl\}$ or finishes at t if $l = fi$ and 0 otherwise.
$\varrho_{\varepsilon e} \in \{0,1\}$	1 if episode e is served with packet ε and 0 otherwise.
$am[\varepsilon] \in \{0,1\}$	The amount of transmitted bits of packet ε.

Our defined problem is modeled as an ILP and then, owing to its complexity, we propose a decomposition using Benders method. Table 5.1 lists all the symbols used in the mathematical formulation. The objective of our problem can be formulated as follows:

$$min \sum_{\substack{p \in \boldsymbol{P} \\ t \leq T}} \tilde{\alpha}_{pt} \,. \tag{OBJ}$$

By words, it is minimizing AoI above the threshold of all the processes throughout the time horizon.

5.4.1 Requirements Constraints

5.4.1.1 Computation requirement

At each computation stage, the virtual machine assigned for each process p should complete the required c_p cycles (the required number of cycles to generate one packet).

$$\begin{aligned} \sum_{m \in \boldsymbol{\mathcal{M}}} \sum_{t=t_1}^{t_2} f_m * \rho_{et}^m * \Delta \geq c_p * (s_e[cp, t_1] \wedge s_e[dl, t_2]) \\ \forall p \in \boldsymbol{P} \;\; \forall e \in \boldsymbol{\mathcal{E}}_p \;\; \forall t_1 \leq t_2 < T_p \,. \end{aligned} \tag{C1}$$

5.4.1.2 Offloading constraints

A packet can not be served unless the previous packet from the same stream is served completely.

$$\begin{aligned} \omega_{\varepsilon t}^r \leq 1 - \omega_{\varepsilon' t'}^r \\ \forall p \in \boldsymbol{P} \;\; \forall s \in \boldsymbol{\mathcal{S}}_p \;\; \forall \varepsilon, \varepsilon' \in \mp_s : \varepsilon \geq \varepsilon' \;\; \forall t, t' : t \leq t' \\ \forall r \in \boldsymbol{\mathcal{R}} \,. \end{aligned} \tag{C2}$$

An episode should be served with one packet from each stream:

$$\sum_{\varepsilon \in \mp_s} \varrho_{\varepsilon e} = 1$$
$$\forall p \in \boldsymbol{P} \;\; \forall e \in \boldsymbol{\mathcal{E}}_p \;\; \forall s \in \boldsymbol{\mathcal{S}}_p \;\; \forall \varepsilon \in \mp_s \,. \tag{C3}$$

A packet should receive enough transmission resources to offload to the edge:

$$am[\varepsilon] = \sum_{t<T} rt(s,t) * \omega^r_{\varepsilon t} \geq u_s$$
$$\forall p \in \boldsymbol{P} \;\; \forall s \in \boldsymbol{\mathcal{S}}_p \;\; \forall \varepsilon \in \mp_s \,. \tag{C4}$$

Now, for each upload stage of a certain process, one packet from each stream among all streams that belong to the process should be transmitted.

$$\sum_{r \in \boldsymbol{\mathcal{R}}} \sum_{t=t_1}^{t_2} rt(s,t) * (\omega^r_{\varepsilon t} \wedge \varrho_{\varepsilon e}) * \Delta \geq$$
$$am[\varepsilon] * (s_e[ul,t_1] \wedge s_e[cp,t_2] \wedge \varrho_{\varepsilon e})$$
$$\forall p \in \boldsymbol{P} \;\; \forall s \in \boldsymbol{\mathcal{S}}_p \;\; \forall \varepsilon \in \mp_s \;\; \forall e \in \boldsymbol{\mathcal{E}}_p \;\; \forall t_1 \leq t_2 < T_p \,. \tag{C5}$$

5.4.1.3 Download constraint

In each download stage, the process should finish broadcasting the output packet to all the destinations.

$$\sum_{r \in \boldsymbol{\mathcal{R}}} \sum_{t=t_1}^{t_2} rt(v(p,t),t) * \omega^r_{et} * \Delta \geq$$
$$d_p * (s_e[dl,t_1] \wedge s_e[fi,t_2])$$
$$\forall p \in \boldsymbol{P} \;\; \forall e \in \boldsymbol{\mathcal{E}}_p \;\; \forall t_1 \leq t_2 < T_p \,. \tag{C6}$$

5.4.1.4 Computation and Streaming orders

An episode can not be processed in a parallel fashion. Meaning, an episode can not be assigned to two virtual machines at the same time unit.

$$\begin{aligned} \sum_{m \in \boldsymbol{\mathcal{M}}} \rho_{et}^{m} \leq 1 \\ \forall e \in \boldsymbol{\mathcal{E}} \;\; \forall t \leq T \,. \end{aligned} \tag{C7}$$

A stream should be served in sequence. A stream can not be served with more than one channel at the same time.

$$\begin{aligned} \sum_{r \in \boldsymbol{\mathcal{R}}} \omega_{xt}^{r} \leq 1 \\ \forall x \in \{\mp \cup \boldsymbol{\mathcal{E}}\} \;\; \forall t \leq T \,. \end{aligned} \tag{C8}$$

5.4.2 Process Structure Constraints

5.4.2.1 Stage order

The stages of episodes should be in order.

$$\begin{aligned} s_e[l_2, t_2] + s_e[l_1, t_1] \leq 1 \\ \forall p \in \boldsymbol{P} \;\; \forall e \in \boldsymbol{\mathcal{E}}_p \;\; \forall t_1 \leq t_2 < T_p \\ \forall l_1 < l_2 \,. \end{aligned} \tag{C9}$$

5.4.2.2 Stage singularity

An episode can have only one upload stage, one computation stage, and one download stage.

$$\sum_{t_1=0}^{T_p}\sum_{t_2=t_1+1}^{T_p} s_e[ul,t] = \sum_{t_1=0}^{T_p}\sum_{t_2=t_1+1}^{T_p} s_e[cp,t] = \sum_{t_1=0}^{T_p}\sum_{t_2=t_1+1}^{T_p} s_e[dl,t] = \sum_{t_1=0}^{T_p}\sum_{t_2=t_1+1}^{T_p} s_e[fi,t] \leq 1 \quad \forall p \in \boldsymbol{P} \;\; \forall e \in \boldsymbol{\mathcal{E}}_p \;\; l \in \{ul, cp, dl\}\,. \tag{C10}$$

5.4.3 AoI Computation Constraints

5.4.3.1 Computation from episode

If episode e finishes at t, AoI of a process at time t is equal to t minus the start time of e.

$$\alpha_{pt} \geq s_e[fi,t] * \sum_{t'<t} s_e[ul,t'] * (t-t') \quad \forall p \in \boldsymbol{P} \;\; \forall t < T_p\,. \tag{C11}$$

5.4.3.2 Computation from previous value

If no episode finishes at time t, AoI at time t is equal to AoI at time $t-1$ plus one.

$$\alpha_{pt} \geq \delta_p[t] * (\alpha_{pt-1} + 1)\,,$$

where: (C12)

$$\delta_p[t] = \bigwedge_{e \in \boldsymbol{\mathcal{E}}_p} \sim s_e[fi,t]$$

$$\forall p \in \boldsymbol{P} \;\; \forall t \leq T_p\,.$$

5.4.3.3 AoI above threshold computation

$\tilde{\alpha}_{pt}$ is equal to α_{pt} minus τ_p at any time. If α_{pt} is less than τ_p, $\tilde{\alpha}_{pt}$ will be zero.

$$\tilde{\alpha}_{pt} \geq \alpha_{pt} - \tau_p$$
$$\forall p \in \boldsymbol{P} \;\; \forall t \leq T_p \,. \tag{C13}$$

5.4.4 Capacity Constraints

5.4.4.1 Computation resources

A virtual machine can not serve more than one process at a time.

$$\sum_{p \in \boldsymbol{P}} \sum_{e \in \boldsymbol{\mathcal{E}}_p} \rho^m_{et} \leq 1$$
$$\forall m \in \boldsymbol{\mathcal{M}} \;\; \forall t \leq T \,. \tag{C14}$$

5.4.4.2 Radio resources

A radio channel can serve only one transmission at a time.

$$\sum_{s \in \boldsymbol{\mathcal{S}}} \sum_{\varepsilon \in \mp_s} \omega^r_{\varepsilon t} + \sum_{p \in P} \sum_{e \in \boldsymbol{\mathcal{E}}_p} \omega^r_{et} \leq 1$$
$$\forall r \in R \;\; \forall t \leq T \,. \tag{C15}$$

5.5 Benders Decomposition

Equipped by proposition 5.1 and the vast number of variables in the ILP, our problem is a very complex problem that requires a scalable technique to attain good solutions. In this section, we propose a logic-based Benders decomposition technique to build a solution evolutionarily. This technique generates a feasible solution in each iteration, a very distinct property among other decomposition techniques. For example, any column-generation-based decomposition [84] must go through a final iteration in order to get a feasible integer solution. Without this final step, the decomposition

method will most probably end up with non-integer solutions. In Benders, the decomposition guarantees the feasibility at each iteration. Hence, whenever the system requires a solution during any iteration, it will get an updated one. When a problem dedicates sets of variables for a specific subproblems only, Benders decomposition takes advantage of that and splits the problem into a master and subproblem/s. The master fixes the values of its variables and sends these values to the subproblems. The subproblems use these values to find their objectives, build a cut (constraint) and send it back to the master in order to update their variables. The algorithm iterates until it gets the optimal solution or until a certain criterion is satisfied (e.g., number of iterations).

Now, our problem requires the accomplishment of three tasks: 1) Resource allocation over the processes; 2) Distributing the resources over the episodes; 3) AoI calculations. All these tasks must respect the limitations of the system resources (e.g., the number of virtual machines). It is clear from table 5.1 and the ILP that the variables set $\boldsymbol{\omega} = \{\omega_{xt}^{r} : x \in \boldsymbol{\mathcal{S}} \cup \boldsymbol{P}, t \leq T, r \in \boldsymbol{\mathcal{R}}\}$ and $\boldsymbol{\rho} = \{\rho_{et}^{m} : e \in \boldsymbol{\mathcal{E}}_p, p \in \boldsymbol{P}, t \leq T, m \in \boldsymbol{\mathcal{M}}\}$ are responsible of assigning the radio and computational resources to the processes. Notice here that we changed the index of the variable $\omega_{\varepsilon t}^{r}$ from packet index ε to the stream index s. This will reduce the number of variables and thus will simplify the master problem. Before running the subproblem, we can find all the packets assignments as they are all sent in order. By fixing these variables values (assuming a solution for the first task), we can split the rest of the problem into several subproblems. Each of these subproblems is responsible for one process. It distributes the assigned resources over the process episodes (task 2) and accordingly calculates the AoI (task 3). Once each subproblem finds a solution for its process, it generates a cut informing the master how good it did by the resources it assigned to the process. With all the cuts accumulated from all the subproblems, the master generates a new

assignment to the resources. As stated earlier, Benders decomposition is an iterative method and will continue iterating until the desired termination criterion is met.

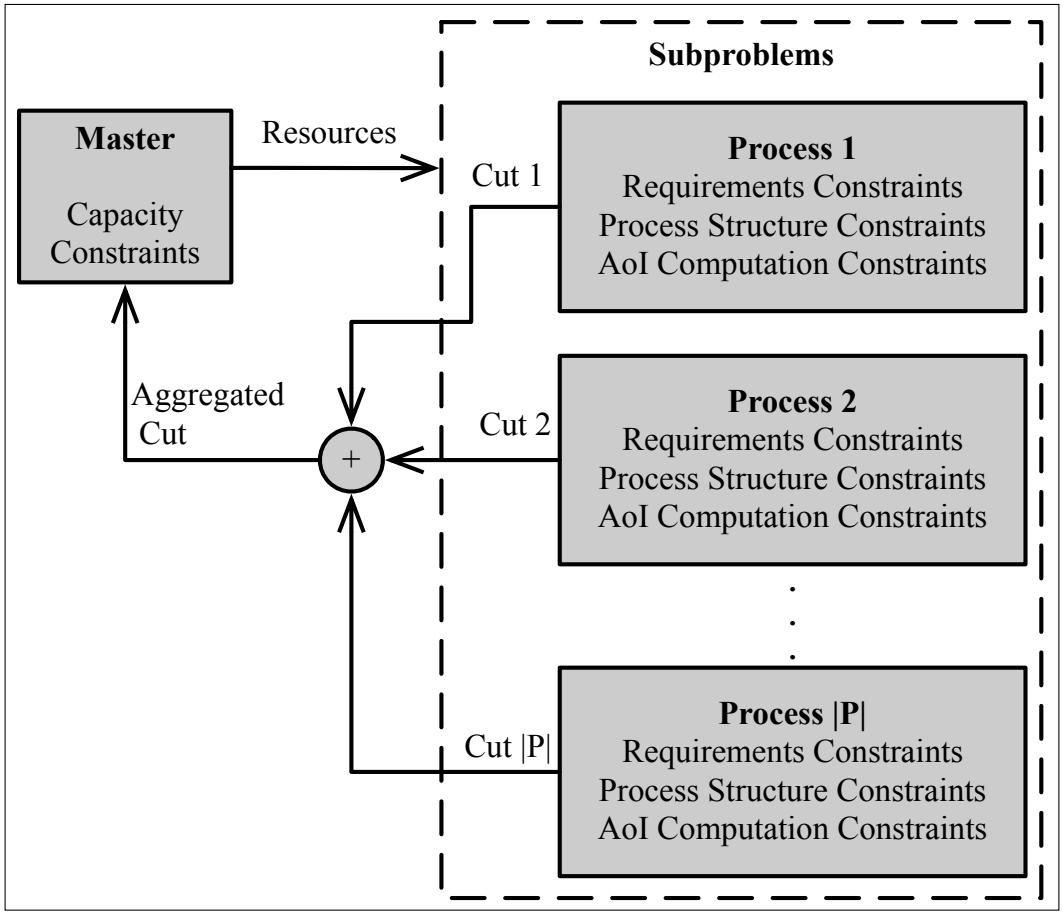


Figure 5.4: Benders Decomposition Scheme

Figure 5.4 schematically describes the overall process of our Benders Decomposition. As the master's role is to assign the resources to the processes, it must respect the capacity of these resources. Hence, as shown in the figure, the capacity constraints reside in the master program. Whereas requirements constraints, the process structure constraints, and the AoI calculation constraints belong to the subproblems. Each subproblem solution must first fit the resources assigned to its process over the episodes (hence the need to the process structure constraints) while considering the requirements of the process (the requirements constraints) then calculates AoI (AoI

calculation constraints).

In the next subsections, we will describe the solution of the subproblems, the cuts generation, and the master solution.

5.5.1 Subproblem Solution

As stated before, each subproblem's role is to utilize the resources assigned to the corresponding process in order to minimize its AoI. The following is the formal definition of the process p subproblem.

Problem S Definition : *Given a process as described in problem* P*, the radio and computational resources assigned to this process depicted by the variables assignments (k is the iteration index)* $\bar{\boldsymbol{\omega}}^k = \{\bar{\omega}_{xt}^{rk} : x \in \boldsymbol{\mathcal{S}} \cup \boldsymbol{P}, t \leq T, r \in \boldsymbol{\mathcal{R}}\}$ *and* $\bar{\boldsymbol{\rho}}^k = \{\bar{\rho}_{et}^{mk} : e \in \boldsymbol{\mathcal{E}}_p, p \in \boldsymbol{P}, t \leq T, m \in \boldsymbol{\mathcal{M}}\}$*, construct the episodes of each process in away that minimizes its AoI above its threshold.*

Our solution to problem S is shown in Algorithm 7.

For each episode e, several possible periods can fit over. The algorithm depicts these possible periods by the discrete interval $\left[t^i..t^f\right]$. To come up with the solution, the algorithm should find a feasible schedule with the minimum AoI. A feasible schedule is one that makes an episode start after its predecessor and finishes before its successor. The pseudo-code iterates over the number of episodes of the process one by one. For each episode, the algorithm checks all the possible periods $\left[t^i..t^f\right]$ to find whether there are enough resources to fit the episode. If it fits, the algorithm finds the best compatible period of the previous episode that leads to the minimum AoI and stores it. Once it goes through all the episodes, the optimal solution to this problem is found.

Besides searching for the minimum AoI, the algorithm tracks the episodes generated in each $\left[t^i..t^f\right]$ period (line 17). In addition, for each tuple $\{e, t^i, t^f\}$, it binds it

Algorithm 7 Solution of the subproblem

```
1: procedure SUBPROBLEM(p, ω̄^k, ρ̄^k)
2:     for e < 𝓔_p do
3:         for t^i ← 0 : T_p do
4:             for t^f ← t^i + 1 : T_p do
5:                 epi = episodeFits(e, ω̄^k, ρ̄^k, t^i, t^f)
6:                 if epi.isScheduled then
7:                     AoI[e, t^i, t^f] ← ∞
8:                     for pt^i ← 0 : t^i do
9:                         for pt^f ← 0 : t^f do
10:                            aoi ← calcAoI(
11:                                    AoI[e − 1, pt^i, pt^f]
12:                                    , t^i, t^f)
13:                            if aoi < AoI[e, t^i, t^f] then
14:                                AoI[e, t^i, t^f] ← aoi
15:                                min_pt^i[e, t^i, t^f] ← pt^i
16:                                min_pt^f[e, t^i, t^f] ← pt^f
17:                                episodes[e, t^i, t^f] ← epi
18:                else
19:                    AoI[e, t^i, t^f] ← AoI[e − 1, pt^i, pt^f]
20:                    min_pt^i[e, t^i, t^f] ← t^i
21:                    min_pt^f[e, t^i, t^f] ← t^f
22:    t^i, t^f ← min_index(AoI[𝓔_p − 1])
23:    e = 𝓔_p − 1
24:    do
25:        schedule.add(episodes[e, t^i, t^f])
26:        t^i, t^f ← min_pt^i[e, t^i, t^f], min_pt^f[e, t^i, t^f]
27:        e = e − 1
28:    while e > 1
29:    return {AoI, schedule}
```

with a period $\left[pt^i..pt^f\right]$ of the previous episode $e-1$ (lines 15-16, 20-21). If the current episode e can fit into the pair $\left[t^i..t^f\right]$, then it stores the period of the previous episode $e-1$ that gives the minimum AoI (lines 8-17). Else, it assigns $\left[pt^i..pt^f\right]$ the values of the current $\left[t^i..t^f\right]$. Finally, the algorithm constructs the solution by backtracking all the way from the last episode $\boldsymbol{\mathcal{E}}_p - 1$ to the first episode (lines 22-28). It finds the pair $\left[t^i..t^f\right]$ with the minimum AoI for the final episode (line 23). Then, from there, it uses the pairs stored $\left[pt^i..pt^f\right]$ for each tuple $\{e, t^i, t^f\}$ to find the previous episode that should be added (see Figure 5.5 for an example).

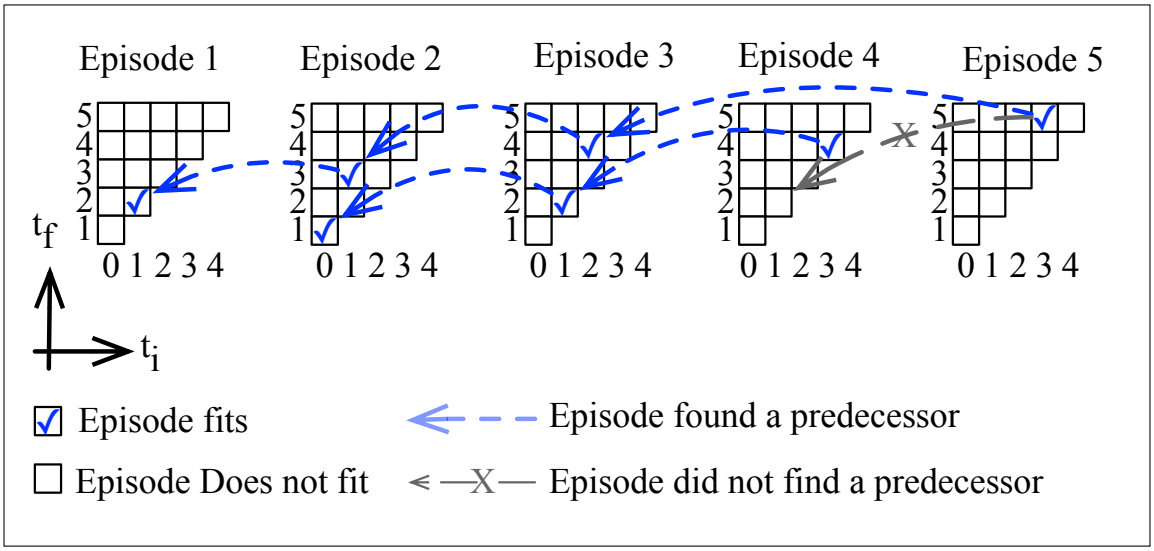


Figure 5.5: Example of a subproblem solution. The example is a process that lasts for 6 time units and has 5 episodes.

Proposition 5.2. *Algorithm 7 solves optimally problem* S *with time complexity* $\mathcal{O}(|\boldsymbol{\mathcal{E}}_p|T_p^3)$.

Proof. We will prove the optimality by induction. When the algorithm starts with the first episode, for each period $\left[t^i..t^f\right]$, it will find the best solution as it will check whether the episode will fit in this period starting from time t^i and ending exactly at t^f. Hence, AoI will be updated correctly. If the episode doesn't fit in the period, the AoI will be set to its maximum value, which means no episodes have been processed. Now, assume the solution is optimum from the first episode to episode e, and let us

see if this will be carried to include episode $e+1$. For any $\left[t^i..t^f\right]$, the algorithm does the same thing as for the first episode except the following: if the episode fits in the period defined by the period, it checks all the solutions of the previous episode to be compatible with the current period (i.e., all the solutions of the previous episodes where it starts before the t^i and finishes before t^f). Then it chooses the one that will end up with the minimum AoI. If the episode does not fit, the algorithm will copy the solution of the previous episode, which is the best feasible solution available for the current pair. This will lead to having all minimum AoI for all possible solutions and get the optimum solution for all possible pairs for all the episodes. Hence the algorithm will end up with the optimal solution. There are $|\boldsymbol{\mathcal{E}}_p|$ episodes and there are $\frac{T_p*(T_p-1)}{2}$ possible number of pairs. The validation function takes a linear time $\mathcal{O}(T_p)$. So the complexity is $\mathcal{O}(|\boldsymbol{\mathcal{E}}_p|T_p^3)$. □

5.5.2 Cut Generation

The cut is a message from a subproblem to the master informing it about the best AoI reached with the resources assigned. This message can be conducted as follows: For each process p, for all its episodes $\boldsymbol{\mathcal{E}}_p$, if an episode e is not scheduled, find the stage that is not being fulfilled and inform the master about it. If the episode is scheduled, inform the master of the possible modifications to the episode that might improve the result.

To formally define a cut, we define the n^{th} *input stream threshold* of the k^{th} iteration as:

$$\sigma_{sn}^{uk} = \sum_{t=t_{sn}^{iuk}}^{t_{sn}^{fuk}} \sum_{r\in\boldsymbol{\mathcal{R}}} \bar{\omega}_{st}^{rk} \times \mathrm{rt}(s,t) \times \Delta\,, \tag{5.5}$$

where $\left[t_{sn}^{iuk}..t_{sn}^{fuk}\right]$ is the n^{th} period of the input stream s. This threshold is the value

that the total assigned resources to stream s in $\left[t_{sn}^{iuk}..t_{sn}^{fuk}\right]$ should be compared with in the next iteration. Also, we can define the n^{th} *computation threshold* as:

$$\sigma_{en}^{ck}(x) = \max\left(\sum_{t=t_{en}^{ick}}^{t_{en}^{fck}} \sum_{m\in\boldsymbol{\mathcal{M}}} f_m \times \bar{\rho}_{et}^{mk} \times \Delta, x\right), \tag{5.6}$$

where $\left[t_{en}^{ick}..t_{en}^{fck}\right]$ is the n^{th} computation period of the episode e and $x \in \{0, c_p - \epsilon\}$ and ϵ is a small value. Similarly, we can define σ_{en}^{dk}, as the *download stream threshold*.

Now, we define the n^{th} *input stream indicator* as:

$$\phi_{sn}^{uk} = \mathbb{1}\left(\sum_{t=t_{sn}^{ik}}^{t_{sn}^{fk}} \sum_{r\in\boldsymbol{\mathcal{R}}} \omega_{st}^{r} \times \mathrm{rt}(s,t) \times \Delta \leq \sigma_{sn}^{uk}\right). \tag{5.7}$$

This indicator will return *TRUE* if the total assigned resources to stream s in $\left[t_{sn}^{iuk}..t_{sn}^{fuk}\right]$ are the same or less than the amount of resource assigned to s in the k$^{\text{th}}$ iteration (see. equation 5.5). Similarly, we define the *computation indicator* ϕ_{en}^{ck} and the *download stream indicator* ϕ_{en}^{dk}. Then, we define the following variable as the *process cut indicator*:

$$\Phi_p^k = \bigwedge_{\substack{s\in\boldsymbol{\mathcal{S}}_p \\ n\leq N_s^{uk}}} \phi_{sn}^{uk} \bigwedge_{\substack{e\in\boldsymbol{\mathcal{E}}_p \\ n\leq N_e^{ck}}} \phi_{en}^{ck} \bigwedge_{\substack{e\in\boldsymbol{\mathcal{E}}_p \\ n\leq N_e^{dk}}} \phi_{en}^{dk}, \tag{5.8}$$

where N_s^{uk}, N_e^{ck} and N_e^{dk} are the number of periods for the input stream s, the computation of episode e and download stream of episode e respectively. In other words, this indicator will return *TRUE* if, in the given periods, 1) the radio resources assigned to every input stream of process p are the same or less than the resources assigned in k$^{\text{th}}$ iteration, 2) the computation resources assigned to each episode are the same or less as k$^{\text{th}}$ iteration and 3) the radio resources assigned to download each episode result are the same or less as the k$^{\text{th}}$ iteration . With this indicator, the cut can be written as follows:

$$z \geq \sum_{p \in \boldsymbol{P}} \zeta_p^k \mathbb{1}(\Phi_p^k)\,, \tag{5.9}$$

where z is the problem objective and ζ_p^k is the process p outcome at the k$^{\text{th}}$ iteration. The cut states that AoI of process p will be the same or greater than AoI of the k$^{\text{th}}$ iteration if the resources assigned to process p in the given periods are the same or less than the resources assigned to p at the k$^{\text{th}}$ iteration. Next, we will explain how to find $\{\phi_{sn}^{uk}\}$, $\{\phi_{en}^{ck}\}$ and $\{\phi_{en}^{dk}\}$.

5.5.2.1 Input Streams Indicators

Unlike the computation requirements and output requirements, the input requirements of the episodes have two distinct properties: 1) An input stream is shared among the episodes. A packet from one input stream can be used by more than one episode, while a computation cycle or an output packet can be used by only one episode. 2) An episode needs only one output stream, needs to be computed once, while it needs to collect data from more than one input stream. These two properties entail that we build the input stream indicators independently from the condition of the episodes (i.e., whether the episode is scheduled or not). For each input stream of a process, we check where a packet has started and where it has completed the transmission and thus build an indicator around this period (call this a *packet indicator*). Then, for each period that does not include any packet transmissions, we construct what is called an *empty indicator*. (see Figure 5.6).

5.5.2.2 Unscheduled Episodes Indicators

After generating the input streams indicators, we check each episode and whether it is scheduled or not. If it is unscheduled, we find the reason for that. The first possible reason is that a packet from a certain input stream was not delivered. A missing

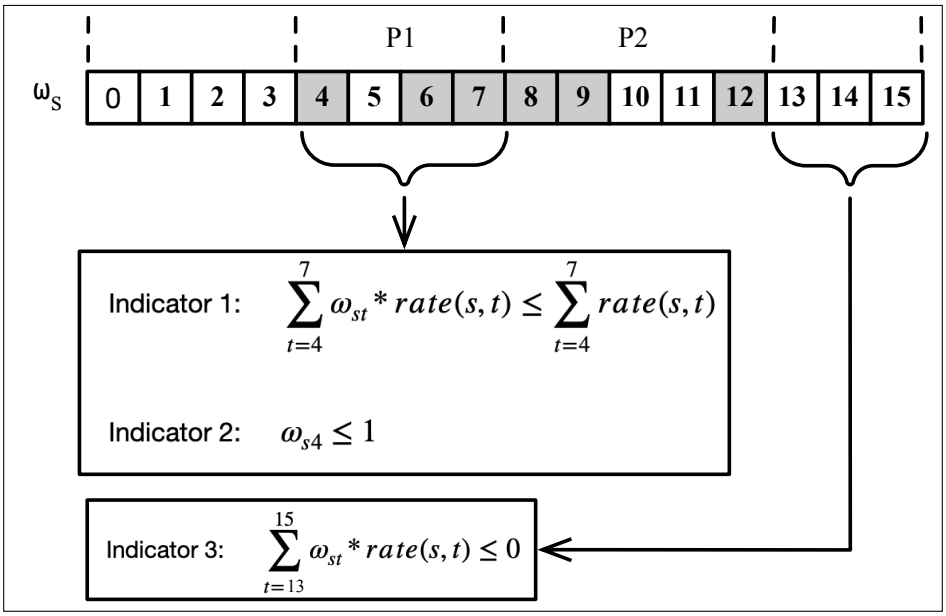


Figure 5.6: Example of input streams indicators with one resource block. These indicators are not all the required indicators.

input stream problem has been handled with the indicators generated in the previous subsection. Otherwise, we start to check the next stage, the computation stage. Now, to find whether this stage is satisfied or not, we need to know when the computation can start. To do so, we have to find all the *computation start candidates (CSC)* time units where an episode can start computing after receiving all the required data. Such a time unit should be after at least one packet transmission from each input stream. Clearly, the first point is where all the streams have delivered one packet. After adding this point, for each subsequently delivered packet from any stream, we add the time unit preceding this packet transmission as a CSC as well. After finding all the candidates, we should examine whether the computation requirement has been fulfilled. If fulfilled, we mark the time unit where the computation has finished as a *download start candidate (DSC)*. If not, we add a computation indicator for the period starting from this candidate until the end of the process duration $\sigma_{en}^{ck}(c_p - \epsilon)$. After that, for each of the time units in the DSC set, we add an indicator starting

from this time unit to the end of the process duration $\phi_{en}^{dk}(d_p - \epsilon)$ (see Figure 5.7).

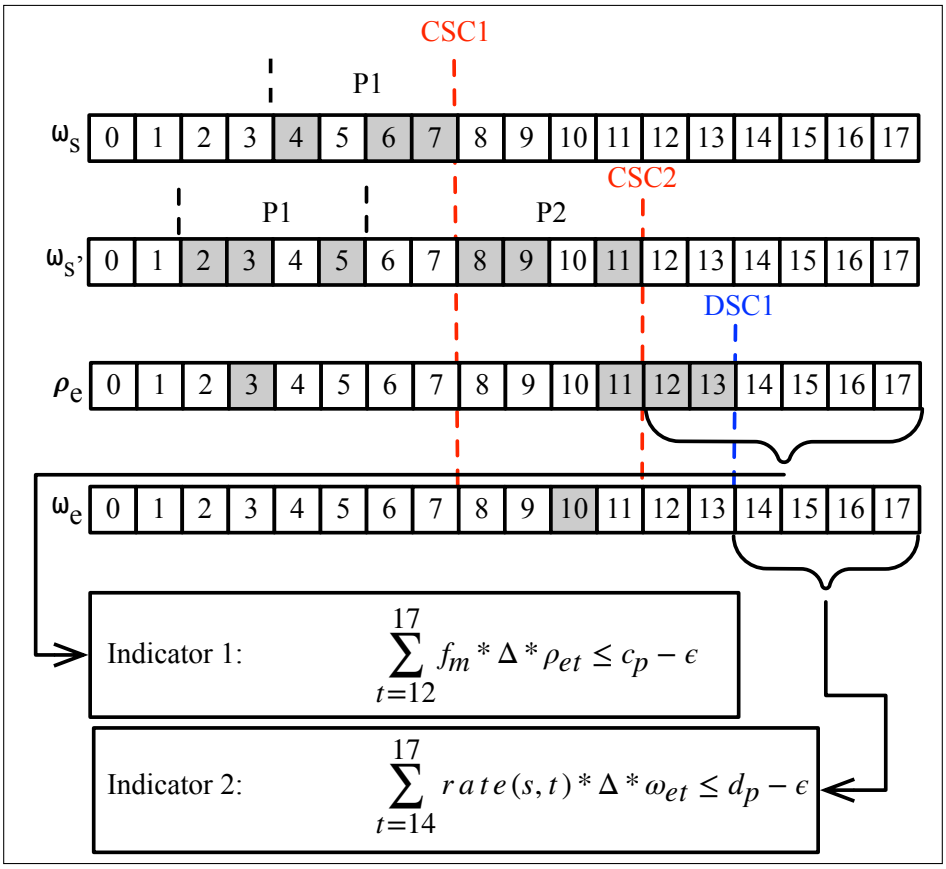


Figure 5.7: Example of unscheduled episode indicators with one resource block and one virtual machine.

5.5.2.3 Scheduled Episodes Indicators

If the episode is scheduled, we need to inform the master how to improve the solution. First, we divide the episode timeline (after the upload stage) into 5 periods: the period between the upload end and computation start (AU), the computation time (C), the period between computation end and the upload start (AC), the download time (D) and after the download end (AD). For period AU, we create the indicator $\phi_{e1}^{ck}(0)$. For period C, we create the indicator $\phi_{e2}^{ck}(0)$. For period AC, we create two indicators: $\phi_{e3}^{ck}(0)$ and $\phi_{e1}^{uk}(0)$. For D and AD respectively, $\phi_{e2}^{uk}(0)$ and $\phi_{e3}^{uk}(0)$ (See Figure 5.8).

Proposition 5.3. *The cut defined in (5.9) is a valid cut for the Benders decomposition.*

Proof. To prove the cut validity, we have to prove that it does not remove feasible solutions from the solution space. In other words, process p schedule that satisfies a process indicator in a k^{th} cut should not have an AoI lower than ζ_p^k. We start with the input streams indicators. Consider the packets indicators. They state that, if the resources assigned to the stream in a given period is the same or less than the currently assigned resources, obviously, the AoI of the process will stay as it is or will get greater than the current AoI as the number of packets will be reduced and an episode requirement might be dissatisfied. If the empty indications stay empty, the packet distribution over the timeline will not change; hence AoI will stay the same. For any periods used to build indicators, if the resources assigned get higher, the distribution of the packets will change and a new cut should be generated.

Regarding the unscheduled episodes, we add indicators for each CSC and DSC points. These indicators state that if the requirement of the episodes (computation and download) are not satisfied, the episode will not be scheduled. If this is the case and given that the schedule of other episodes is the same, the AoI of the process will remain the same. The small number ϵ is used here to ensure that the amount of resource assigned by the master resource did not achieve the required amount.

For scheduled episodes, for the computation period and the download period, we add indicators that state, if the amount of resources assigned to the episode remains the same, the process AoI will remain the same. And if it gets less, AoI might get higher. This is true since, with lower resources, the requirement might not be fulfilled. For the period between the computation and download stage, we add indicators that state that if the higher resources were assigned for either stage (computation or download), AoI might change as both stages might change, and

hence AoI will change. This also applies to the period between the upload period and the computation period, but we only added an indicator for the computation stage since the upload stage was handled with the input streams indicators. The only remaining indicators are two indicators. The indicator of the first time unit of the packets periods and the last time unit of the download period. These two indicators guarantee that the episode starts and ends at these specific time units. Otherwise, AoI might be less with different episode start or end. □

5.5.3 Master Problem Solution

The formal definition of the master problem is:

Problem M Definition : *given a set of virtual machines $m \in M$ and set of resource blocks $r \in R$, each with its own capacity, and given set of cuts in the form defined in (5.9), find an assignment for R and M that minimizes the average AoI of all the processes.*

Proposition 5.4. *Problem* M *is NP-Hard.*

Proof. Consider the well-known NP-Hard problem $M|r, pmtn| \sum w_i U_i$ where the solution should assign jobs $j \in J$ to parallel machines $q \in Q$. Each task has a weight w_j, release time r_j, deadline d_j and processing time h_j. The objective is to maximize the number of admitted tasks. We can reduce this problem to problem M as follows: For each machine q, create a virtual machine m with the same capacity. Create $|J|$ number of resource blocks $r \in R$. For each job j, create a process p with one episode and assign upload and download packets size that would take only a one time unit in the resource blocks. Assign process p computation cycles equal to the processing time h_j. Let $\zeta_p = w_j$ where p is the process corresponding to job j. The cut should force the computation of p to finish after the release time r_j and before the deadline

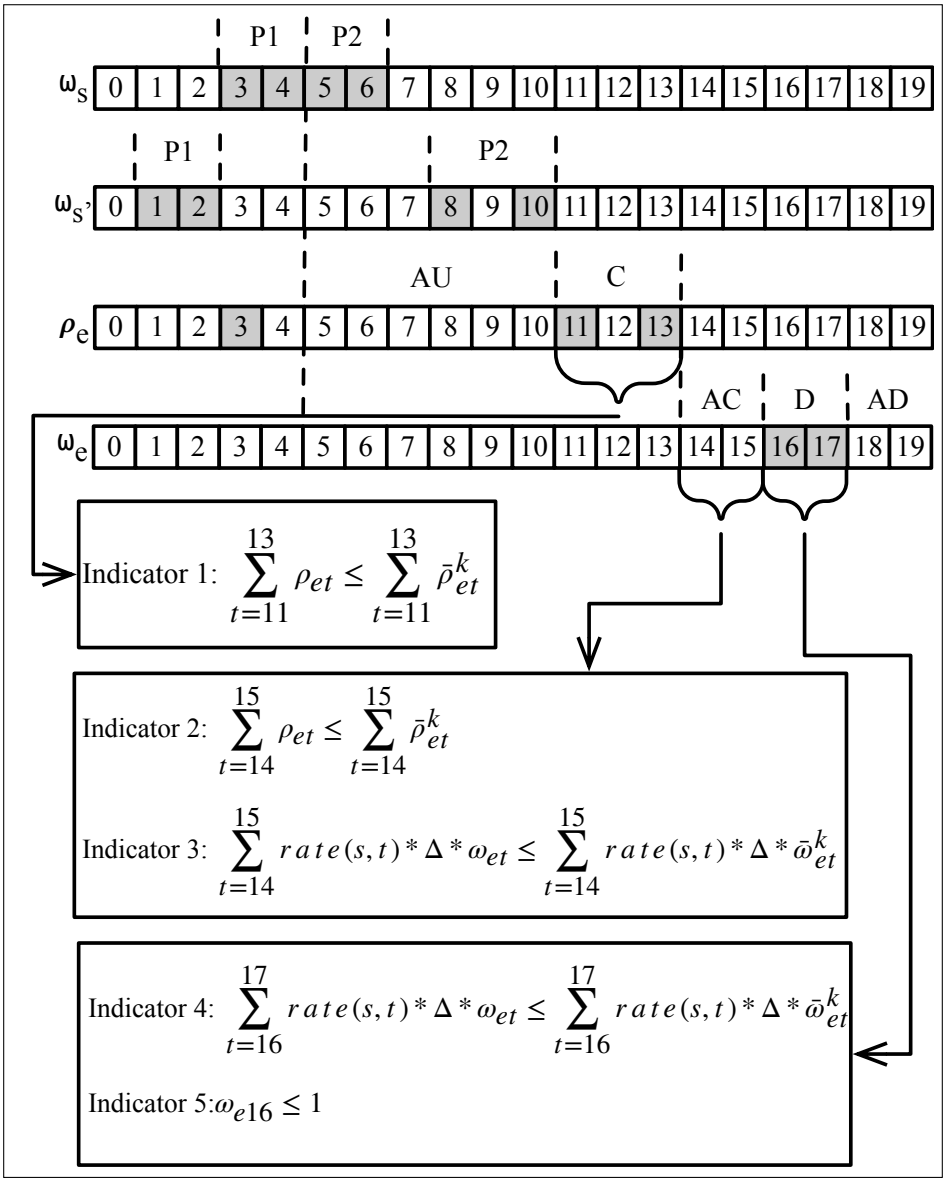


Figure 5.8: Example of scheduled episode indicators with one resource block and one virtual machine.

d_j. This can be done by creating an indicator with $t^{ick}_{en} = r_j$ and $t^{fck}_{en} = d_j$ for the computation requirement of the process of job j. For each process, there is only one indicator $\phi^{ck}_{en}(p_j)$. By that, a solution will always try to minimize the AoI by fulfilling processes. The lower the sum of the AoI of processes that were fulfilled, the better the solution. This will produce a solution for the original problem as each process corresponds to a job. By that, we reduced an NP-hard problem to M in polynomial time. Hence, M is NP-hard. □

The following is the ILP of the master problem:

$$\min z\,.$$

$$s.t.$$

$$\sum_{p\in P}\sum_{e\in Ep}\rho^m_{et} \leq 1 \quad \forall m \in \boldsymbol{\mathcal{M}} \quad \forall t \leq T\,. \tag{MC1}$$

$$\sum_{s\in\boldsymbol{\mathcal{S}}}\omega^r_{st} + \sum_{p\in\boldsymbol{P}}\sum_{e\in\boldsymbol{\mathcal{E}}_p}\omega^r_{et} \leq 1 \quad \forall r \in \boldsymbol{\mathcal{R}} \quad \forall t \leq T\,. \tag{MC2}$$

$$z \geq \sum_{p\in\boldsymbol{P}}\zeta^k_p \mathbb{1}(\Phi^k_p) \quad \forall k \in K\,. \tag{Cuts}$$

$$\rho^m_{et} \in \{0,1\} \quad \forall e \in \boldsymbol{\mathcal{E}}_p \quad \forall p \in \boldsymbol{P} \quad \forall t \leq T_p \quad \forall m \in \boldsymbol{\mathcal{M}}\,.$$

$$\omega^r_{st} \in \{0,1\} \quad \forall s \in \boldsymbol{\mathcal{S}}_p \quad \forall p \in \boldsymbol{P} \quad \forall t \leq T_p \quad \forall r \in \boldsymbol{\mathcal{R}}\,.$$

Constraints MC1 and MC2 are the capacity constraints of the original ILP of the problem. Constraints Cuts are the cuts added in each iteration from the subproblems solutions. As the master problem is NP-hard, we will provide a heuristic method to solve the problem, which is shown in Algorithm 8.

To explain algorithm 8, we need first to explain the function $factor_k(stream, t)$.

Algorithm 8 Master Solution

```
1: procedure MASTER(VMs, RBs, cuts)
2:     for t ≤ T do
3:         for p ∈ P do
4:             for is ∈ p.inputStreams do
5:                 value ← 0
6:                 for cut ∈ cuts do
7:                     value ← value+
8:                             cut.factor_k(is, t)
9:                 sortedStructure[t].
10:                            insert(is, value)
11:            for e ∈ ℰ_p do
12:                os = p.outputStream(e)
13:                value ← 0
14:                for cut ∈ cuts do
15:                    value ← value+
16:                            cut.factor_k(os, t)
17:                sortedStructure[t].
18:                            insert(os, value)
19:        fillBlocks(RBs, sortedStructure[t], t)
20:    for t ≤ T do
21:        for p ∈ P do
22:            for e ∈ ℰ_p do
23:                value ← 0
24:                for cut ∈ cuts do
25:                    value ← value+
26:                            cuts.factor_k(e, t)
27:                sortedStructure[t].insert(e, value)
28:        fillMachines(VMs, sortedStructure[t], t)
```

This function calculates the importance of assigning a resource block to *stream* at time t. Also, $factor_k(episode, t)$ calculates the importance of assigning a virtual machine to *episode* at time t. As the role of the master in Benders decomposition is to explore new solutions that has not been tested yet (i.e., solutions that do not satisfy any current cut indicator), the idea of the heuristic is to find assignments that can ***violate*** all or most of the cuts. Now, to violate a process indicator, an assignment should violate one of the stages indicators. The contribution of such a violation can be quantified by finding the ratio between the amount of resources taken by such an assignment to the threshold of the indicator. Consider the following example cut:

$$\begin{aligned} z \geq 10 * \Big[& \mathbb{1}(0.1 * \omega^1_{s0} + 0.2 * \omega^1_{s1} \leq 0.2) && \wedge \\ & \mathbb{1}(0.2 * \rho^1_{e2} + 0.2 * \rho^1_{e3} \leq 0.1) && \wedge \\ & \mathbb{1}(0.4 * \omega^1_{e4} + 0.01 * \omega^1_{e5} \leq 0.01) \Big] && + \\ 5 * \Big[& \mathbb{1}(0.1 * \omega^1_{s0} + 0.2 * \omega^1_{s1} \leq 0.2) && \wedge \\ & \mathbb{1}(0.2 * \rho^1_{(e+1)2} + 0.2 * \rho^1_{(e+1)3} \leq 0.1) && \wedge \\ & \mathbb{1}(0.3 * \omega^1_{(e+1)4} + 0.05 * \omega^1_{(e+1)5} \leq 0) \Big] \end{aligned} \tag{5.10}$$

This shows an example cut for two processes, each with one episode and one input stream. For simplicity, the example assumes there is only one resource block and one virtual machine. It is obvious here that the contribution of assigning the resource block to stream s at time slot 1 is equal to the portion that the assignment will take from the indicator threshold (in this case, the threshold is 0.2). Hence, the gain is equal to 0.5 (0.1/0.2). In addition, violating this indicator will reduce the lower bound of the objective by 10. Thus, we propose the equation 5.11 to calculate the factor. Let $\langle s', t\rangle \in \phi^{uk}_{sn}$ if $s' = s$ and $t^{ick}_{en} \leq t \leq t^{fck}_{en}$. Let I^{uk}_p be the set of input streams indicators that are part of the process indicator Φ^k_p, and $\sigma(\phi)$ be the threshold of indicator ϕ. Then, the factor is given by:

$$factor_k(s,t) = \sum_p \frac{\zeta_p^k}{T_p} g_k(s,t) \tag{5.11}$$

$$g_k(s,t) = \begin{cases} 0 & \text{if } \forall \phi \in I_p^{uk} \ \langle s,t \rangle \notin \phi \\ min\{\max_{\langle s,t \rangle \in \phi} \frac{rate(s,t) \times \Delta}{\sigma(\phi)}, 1\} & \text{otherwise} \end{cases} \tag{5.12}$$

Clearly, an assignment of the resource block to the pair $\langle s,t \rangle$ will not violate an indicator ϕ unless $\langle s,t \rangle \in \phi$. That is why in the first case of equation (5.12), we set the gain to zero. In the second case, the equation assigns the gain of the pair $\langle s,t \rangle$ the highest ratio among all the indicators that satisfy ϕ unless $\langle s,t \rangle \in \phi$. That's, of course, if it is less than 1 (an assignment can not violate an indicator more than once). We only consider the highest contribution as it is enough to emphasize the importance of such an assignment.

Since an input stream belongs to only one process, the sum in (5.11) will end up with only one non-zero term. Notice that we divide the term with the duration of the process for normalization purposes. Without this normalization, a process with a longer duration will have an unnecessary advantage.

Similar to $factor_k(s,t)$ explanation, we can provide an explanation for function $factor_k(episode,t)$. We omit that for the sake of brevity.

Now that we have explained the factor term, it is much easier to demonstrate Algorithm 8. The algorithm in lines (2-18) goes through the input streams and download streams, for each time unit, and ranks them according to their total cuts factor ($value$). The ranking is done by inserting the pair $\langle stream, value \rangle$ into a sorted data structure (i.e., red-black tree). In line 19, the algorithm calls the function $fillRBs$ which fills the available resource blocks at a certain time unit t with the streams sorted in the data structure. Similar to the logic of filling the resource

blocks, the virtual machines are filled with computation requirements of the episodes following the same step in lines (20-28).

Proposition 5.5. *Algorithm 8 finds a feasible solution for the master problem with time complexity* $\mathcal{O}(T \times |C| \times ((|E| + |S|))$ *where* $\boldsymbol{\mathcal{E}} = \bigcup_p \boldsymbol{\mathcal{E}}_p$ *and* $S = \bigcup_p S_p$.

Proof. Regarding feasibility, the functions $fillRBs$ and $fillVMs$ respect the capacity of the resource blocks and virtual machines respectively. Since for each time unit, a stream or episode are considered only once, the algorithms respects also the requirement that the transmission of streams and the computation of episodes should be accomplished sequentially. The complexity of factor calculation is constant $\mathcal{O}(1)$. So the complexity of the *for* loops in lines 4-10, 11-18 and 22-27 are $\mathcal{O}(|\boldsymbol{\mathcal{S}}_p| \times |\boldsymbol{\mathcal{C}}_| + \log |\boldsymbol{\mathcal{S}}_p|)$, $\mathcal{O}(|\boldsymbol{\mathcal{E}}_p| \times |\boldsymbol{\mathcal{C}}_| + \log |\boldsymbol{\mathcal{E}}_p|)$ and $\mathcal{O}(|\boldsymbol{\mathcal{E}}_p| \times |\boldsymbol{\mathcal{C}}_| + \log |\boldsymbol{\mathcal{S}}_p|)$ respectively. Filling the resources takes a linear time so the time complexities of lines 19 and 28 are $\mathcal{O}(|\boldsymbol{\mathcal{E}}_p| + |\boldsymbol{\mathcal{S}}_p|)$ and $\mathcal{O}(|\boldsymbol{\mathcal{E}}_p|)$ respectively. So the overall time complexities of the for loops in 1-19 and 20-28 are $\mathcal{O}(T \times ((|\boldsymbol{\mathcal{E}}| + |\boldsymbol{\mathcal{S}}|) \times |\boldsymbol{\mathcal{C}}| + \log(|\boldsymbol{\mathcal{E}}|) + \log(|\boldsymbol{\mathcal{S}}|)) + |\boldsymbol{\mathcal{S}}| + |\boldsymbol{\mathcal{E}}|)$ and $\mathcal{O}(T \times (|\boldsymbol{\mathcal{E}}| \times |\boldsymbol{\mathcal{C}}| + \log(|\boldsymbol{\mathcal{E}}|)) + |\boldsymbol{\mathcal{E}}|)$ respectively. After neglecting minor terms, we end up with $\mathcal{O}(T \times |\boldsymbol{\mathcal{C}}| \times ((|\boldsymbol{\mathcal{E}}| + |\boldsymbol{\mathcal{S}}|))$ □

5.6 Performance Evaluation

We evaluate the performance of the proposed methods, including Mixed Integer Linear Programming (MILP), ILP-Master Benders (IMB) decomposition (the master is solved via CPLEX branch-and-cut), and our Heuristic-Master Benders (HMB) decomposition; we consider two performance metrics: execution time and average AoI. We then evaluate the system's performance (utilizing HMB) by using computation load, input stream packet size, number of input streams, and AoI threshold. We trace the vehicular traffic using the well-known traffic simulator SUMO (see Appendix B.1).

Table 5.2 shows the parameters used throughout this section unless mentioned otherwise. We assume an RSU with $500m$ coverage range. We use CPLEX to solve our optimization models and C++ to simulate the operation of our algorithms. We generate results on CPU with Intel(R) Core(TM) i7-6700 CPU @ 2.7GHz, 16GB memory ram, and 64-bit mac operating system. The results are averaged over ten runs.

Table 5.2: Used Parameters

Factors	**Distribution**	**Mean**	**Variance**
Number Of Virtual Machines	-	4	-
Virtual Machines Speed (GHz)	Gaussian	2	0.2
Req. Number of Cycles (10^6)	Gaussian	1	0.1
Vehicle's Velocity	Trunc. Gaussian	25	7
Number Of Resource Blocks	-	6	-
Resource Block Bandwidth (MHz)	-	20	-
Upload Data Size (MB)	Gaussian	0.1	0.1
Download Data Size (MB)	Gaussian	0.1	0.1
Input Streams per Process	-	2	-
Processes Time Horizon (ms)	-	30	-

5.6.1 Scheduling Performance

5.6.1.1 Methods Comparison

We analyze and evaluate the performance of our proposed HMB decomposition method. First, we study the scheduling performance of our method versus the optimal solution obtained from the MILP and IMB decomposition by considering the execution

time and average AoI. As the number of iterations required to converge to the optimal solution is exponential, we limit the number of iterations for IMB and HMB decomposition methods to 200.

Table 5.3 shows the execution time of the three methods by varying the number of processes and episodes as input instances of the problem. Each process requires two input streams (from two different vehicles) for all the instances generated and sends the result to two different ones. The number of time units required for each stage is set to be one on average; the reason is to give the ILP and IMB decomposition a chance to solve the problem in a reasonable amount of time. However, in the next subsection, we vary this term to evaluate even further the performance of the proposed algorithm. The time horizon is set to 20 time units. From the table, we can observe that, in terms of execution time, even for a small size input instance, the HMB decomposition method surpasses the optimal solution obtained by MILP and IMB (except for the case where we have one process and one episode). For instance, the MILP takes more than two hours to solve for 1 process and 3 episodes, whereas it fails to find a solution after executing for several days for larger instances. Such computation time is not acceptable for a real scenario. Whereas, HMB method maintains an acceptable computational time. Compared to IMB, HMB maintains an acceptable performance in terms of the average AoI. For instance, the highest gap difference is about 20% for the case where the numbers of processes and episodes are two each. While in some other cases, the HMB method surpasses the IMB method(for example, when the numbers of processes and episodes are 3 each, the HMB method performs 12% better than IMB). In terms of speed, HMB was faster with all the problem sizes reported. The best case is for the smallest size (92% speedup) and for the biggest size, the speedup is 88%. As the number of cuts increases in each iteration, the time of both methods increases as well. However, the runtime of IMB increases

exponentially since the branch-and-cut method implemented by CPLEX runs with exponential complexity in terms of the number of constraints.

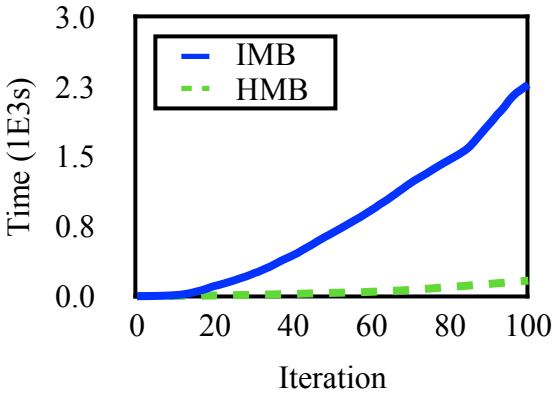
In general, the master of a Benders-decomposition role is to explore the solution space to find an assignment of the variables that minimizes (in case of minimization problem) the objective. But, when the solution space size is large and there are many variables involved in the cut, the master will find several solutions with the same master problem minimum objective (probably zero in our problem). Also, an ILP master solution usually seeks to find all the valid cuts. Whether these cuts will help to find a good solution quickly or not is not of its concern. This is manifested by the fact that IMB does not utilize all the available resources (virtual machines or resource blocks) in its early iterations. Whereas HMB fully utilizes all the available resources in each iteration. Although this will make its convergence harder, it will indeed help to find a better solution in fewer iterations. From table 5.3, we already noticed that interchangeably, IMB and HMB surpass each other in terms of the average AoI.

Table 5.3: Comparison between the performance of algorithms in terms of computation time (ms) and AAoI.

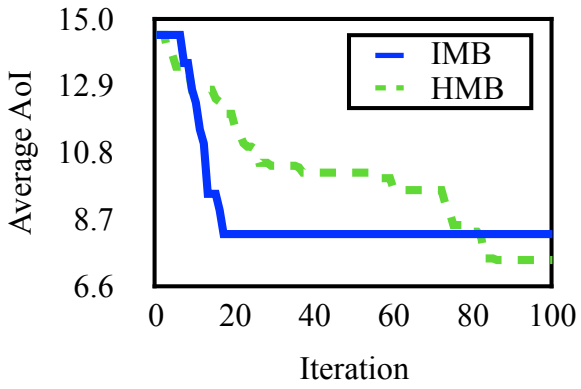
$\|\boldsymbol{P}\|$	$\|\mathcal{E}_{\mathbf{p}}\|$	**ILP**		**IMB**		**HMB**	
		Time	**AAoI**	**Time**	**AAoI**	**Time**	**AAoI**
1	**1**	1885	5.9	90455	5.9	4308	6.3
1	**2**	94708	4.7	142838	4.7	8186	4.9
1	**3**	1.2E7	4.1	203028	4.1	10739	4.5
1	**4**	-	-	300272	4	21683	4.2
2	**2**	-	-	233733	5.2	23592	6.3
2	**3**	-	-	331203	5.0	36577	4.7
2	**4**	-	-	428685	5.4	46647	4.8
3	**3**	-	-	486120	6.1	59220	5.3
3	**4**	-	-	509299	6.0	60728	6.5

5.6.1.2 Evaluation of Benders method

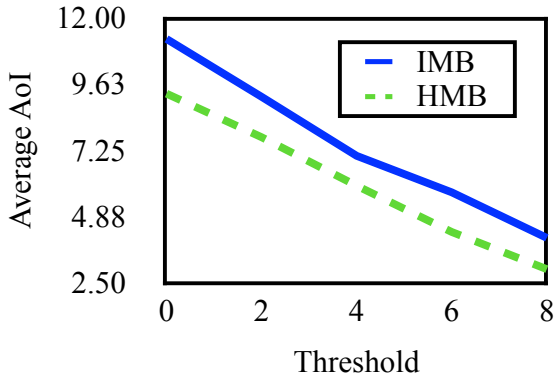
Figures 5.9 and 5.10 illustrate the performance of the proposed methods in terms of the computation time and average AoI, respectively, by varying the number of iterations required for the Benders decomposition. The numbers of processes and episodes are set by 4 each, and the time frame is fixed by 30 time units. As shown in Figure 5.9.b, both methods (IMB and HMB) as expected obtain better results as the number of iterations increases. However, after 82 iterations, the HMB method performs better, as explained earlier in this section.



(5.9.a) The time elapsed vs. the number of iterations.



(5.9.b) The average AoI reached vs. the number of iterations.



(5.9.c) Average AoI vs. process AoI threshold

Figure 5.9: IMB VS HMB

5.6.1.3 AoI above Threshold

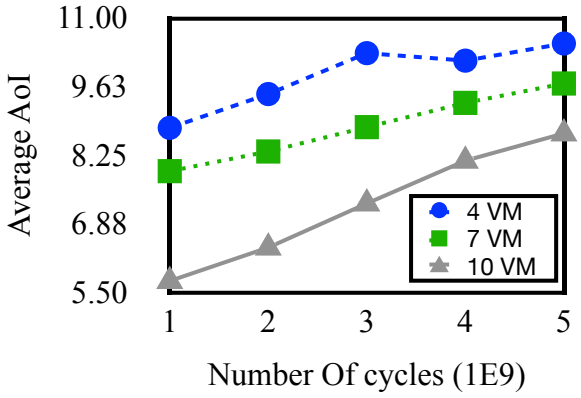
We study the performance of the two Benders decomposition methods by varying the AoI threshold of the processes. Figure 5.9.c reports the average AoI above a certain threshold. As noticed from Figure 5.9.c, HMB always outperforms IMB with a higher threshold. The highest percentile difference is when the threshold is 8 (27%), and the lowest is when the threshold is 4 (16%).

From the analysis conducted, we conclude that the HMB method is the more suitable choice in terms of both execution time and average AoI. Hence, in the next section, we rely on the HMB method to evaluate the performance of our system.

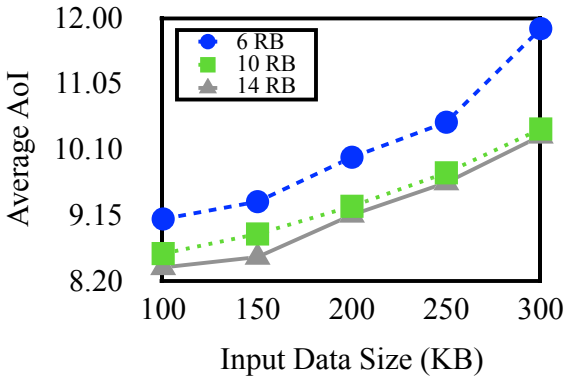
5.6.2 Computation Requirement

In this subsection, we study the impact of increasing the number of required computational server cycles for processes. Figure 5.10.a shows the average AoI trend while increasing the average number of required cycles. Here, we evaluate the system's performance for different numbers of virtual machines (4, 7, and 10), each with 2.0 GHz speed. As expected, the average AoI for all different servers increases as the number of required cycles to execute processes increases. For example, with a server with 7 virtual machines, the average AoI increases in average 5% every 1 million cycles and 11% in the case of 10 virtual machines. From the figure, we can also notice that the system's performance obviously improves by increasing the number of virtual machines. For instance, we have an average of 11% improvement when we use 7 VMs in place of 4 VMs, and 18% improvement when we use 10 VMs in place of 7 VMs. In addition, the figure also shows that the improvement decreases by increasing the load. For example, the improvement from 7 VM to 10 VM is 28% with 1 million cycles and 10% only with 5 million cycles. This means that for a large load, increasing the number of virtual machines will have a minimal impact due to the difficulty of

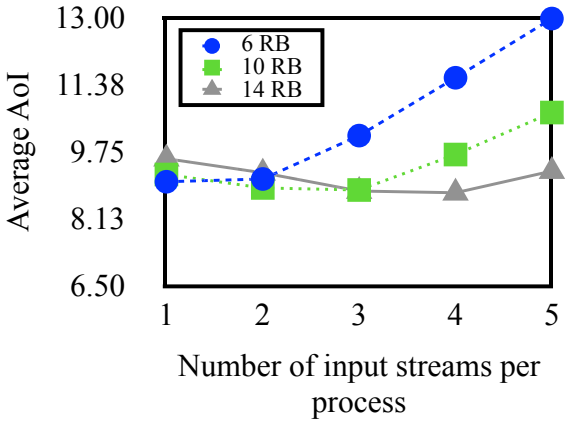
utilizing such a capacity for a heavy load.



(5.10.a) Average AoI vs. expected number of cycles.



(5.10.b) Average AoI vs. data size



(5.10.c) Average AoI vs. load

Figure 5.10: HMB Performance.

5.6.3 Input Data Size

Figure 5.10.b shows the impact of increasing the input data size on the performance of the server. It is clear that as the size of offloaded packet increases, the average AoI increases as well. This is because more time is then required to transmit one packet for each data stream. For example, when there are 10 resource blocks, the average AoI increases on average by 6.9% in the case of 6 resource blocks and 4.9% in the case of 10 resource blocks. The figure also shows that when there are more radio resources

available to transmit data streams, the better the system performs. On average, with 6 resource blocks, the average AoI increases with 7% every 50 KB increase in the input data size, and 5% with 10 resource blocks. The system's performance improvement from 10 resource blocks to 14 resource blocks is quite small (only 4% in the best case). That is because 10 resource blocks can handle the transmission of a large input data size.

Figure 5.10.c shows the impact of increasing the number of input streams per process. It is quite interesting to see that, for few input streams, the less number of resource blocks performs almost the same. This is due to the following reason: data from a single source can only be sent in sequence. Increasing the number of resource blocks will not make a difference. All trends shown reached some point where the average AoI started to increase as the resource block has been fully utilized and the number of episodes scheduled decreased. In average, the increase of the average AoI is 9.6%, 4.3% and 0% for 6, 10, and 14 resource blocks respectively.

5.7 Conclusion

The employment of edge computing in a vehicular environment has several substantial benefits, including the support and assistance of cooperative autonomous driving. Such support provides vehicles with computational capabilities, reduces the number of consumed radio resources, and synchronizes vehicles' actions while accomplishing on-the-road tasks. This work studied the impact of such employment by assuming that RSUs are deployed with edge servers that support the vehicles by receiving data streams from them, processing and sending back to each vehicle the required action to perform the task needed. We mathematically modeled the joint problem of scheduling transmissions and assigning computing resources to minimize AoI above the threshold of CAD processes. The scheduling problem was formulated as an ILP and decomposed

through Benders method. We presented an optimal polynomial-time solution for the subproblem and a heuristic to solve the master problem. The suggested solution showed high scalability to solve the problem compared with the off-the-shelf branch-and-cut algorithm implemented by CPLEX. The problem showed a high intractability that the state-of-the-art CPLEX branch-and-cut failed to solve. On the other hand, although we solved the master problem with a heuristic, with the given number of iterations, the overall performance of the proposed solution surpassed in terms of efficiency both the branch-and-cut approach and benders with ILP-based master solution. Further, our solution showed robustness against increasing the resources required (both radio and computation).

Chapter 6

Conclusion and Future Research Directions

This chapter concludes the presented thesis and highlights future research directions.

6.1 Conclusion

The realization of fully intelligent transportation systems is imminent. As 5G telecommunication technologies, the internet of things, and autonomous driving paradigms all interact, under the ITS umbrella, they provide smooth road traffic and safer and faster trips to everyone. As a subtle solution to many reliability and latency issues of its predecessors, Edge computing plays a significant role in expediting the system components' computability and assisting their interactions. This thesis demonstrates several solutions that should be part of vehicular edge computing to support ITS and aid autonomous driving efficiently.

We started by studying the impact of edge computing in improving computational capabilities. We assumed that 5G communication technology was available on both vehicles and RSU, and the vehicles had sufficient wireless bandwidth to submit time-intensive tasks to the edge. In such a scenario, it was enough to solve the problem of allocating the computational tasks offloaded by the vehicle over the edge servers. We model the problem as an ILP and solve it via Lagrangian relaxation. The outcome was comparable to the optimal solution in terms of the objective and 95% faster than branch-and-cut. We expanded the idea of utilizing the edge server to harvest the computational resources available in the vehicle OBUs. Here, we jointly optimize the computational and the wireless resources to maximize the number of admitted tasks requested by the cars, the IoT devices, and the pedestrian to offload over the vehicles' OBUs and the edge server. We decomposed the problem via the Dantzing-Wolfe technique and solved the resulted problems with polynomial-exact solutions. Compared to CPLEX performance, our solution showed high scalability and reached

a near-optimal objective. Then we decided to embrace a new metric, AoI, that characterizes the system performance by the recency of the information shared and studied the meaning of optimizing such a metric in a vehicular network. We utilize their new technology, RIS, to improve each source-destination vehicle pair's wireless channel condition. Last, we proposed a more specific computational model where the workload is specified by a process that continuously receives information from different sources to develop decisions that support on-the-road activities. The goal was to optimize the AoI of the edge server's decisions computed and delivered to the destination vehicles. We developed a solution based on Benders decomposition and, compared to CPLEX, it showed a high speed in finding efficient solutions.

6.2 Future Research Directions

MEC-assisted CAD is a paradigm that will shift up the performance of AVs on the street. In this thesis, we proposed schedulers that handle the core problem in such a paradigm: resource allocation over the offloaded workload. In our future work, we will try to broach several other critical aspects for any ultra-reliable-low-latency applications such as CAD. Reliability, fault-tolerance, and adaptability are all concerns that we should provide sophisticated solutions to provide the required quality of service level (e.g., 99.99% service reliability). To be more particular, our future targets will be the following:

1. **Fault Tolerance of the edge server**: as part of the design of any system, fault tolerance is an important aspect that requires a separate requirement study, especially in the case of URLLC applications. A failure in data transmission or computation might lead to fatal consequences. We will study how to avoid such a failure and, in case it occurs, how to mitigate/prevent the results. To do

so, we will accomplish the following:

(a) We will design/implement an agent that evaluates the overall environment's situation and accordingly configure the system parameters to avoid such a failure. We can provide that via supervised machine learning approaches.

(b) We will propose specific protocols for each failure type and environment condition to allow the vehicles to abort the road action safely in case of failure. Such protocols require the implementation of statistical models to represent the system condition to provide a sequence of actions taken by the vehicles to lower the probability of casualties or injuries.

2. **Long-term optimization**: Vehicular network is a very dynamic system. Its environment changes rapidly due to weather conditions, the vehicles' arrival flow, the car's paths, and many other factors. In this work, we will investigate the possibility of reconfiguring the system to adapt to these changes to optimize the outcome of the system in a long-term fashion. We can accomplish this through semi-supervised learning approaches.

3. **Data-driven solutions**: As an extension to this thesis work, we will study directly the characteristics of the computational workload a vehicular network might demand. Such a study will allow us to design an edge computing server for these specific computational tasks, improving its performance excessively.

4. **RIS Assisted CAD**: We will investigate the adoption of an intriguing technology called reconfigurable intelligent surface (RIS) to optimize AoI in vehicular networks. RIS is an agile technology that got the researchers' attention, particularly in wireless communication. It provides a solution to wireless channel blockage and signal vanishing. In this research contribution, we will explore the

possibility of utilizing this technology in vehicular networks. We will provide a scheduler that allocates radio resources to serve the data stream flow between vehicles and configure RISs deployed on the sides of the street (see figure 6.1). Such allocation will tackle two crucial problems that vehicular networks always confront; 1) the communication blockage and 2) the signal vanishing. Both issues are magnified when we utilize a short-wave spectrum such as mmWave and Terahertz spectrums. Communications that occur over these spectrums require a pure line of sight between the source and the destination.

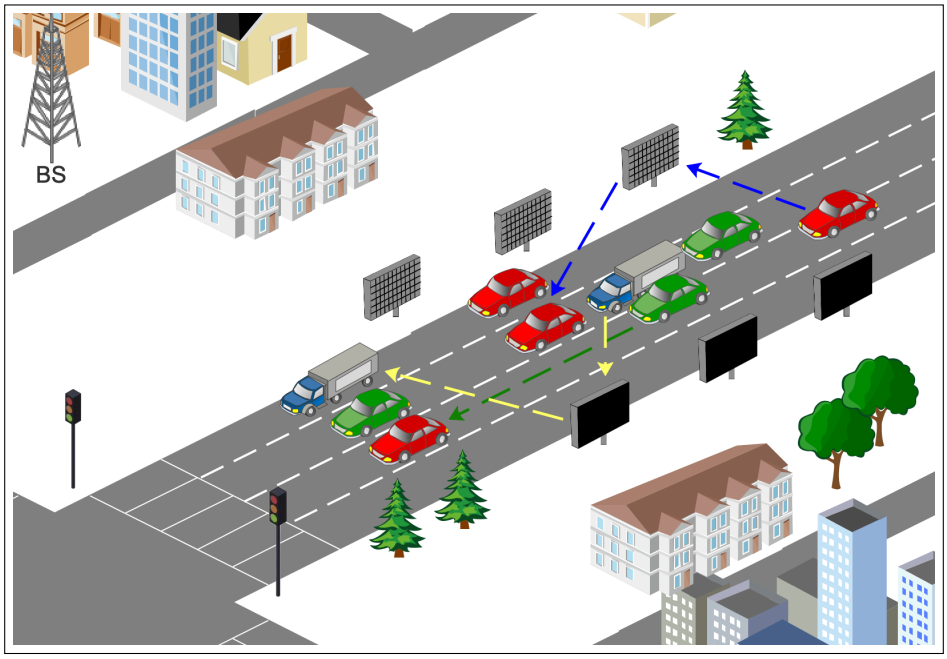


Figure 6.1: The system of model example that adopts RIS technology.

Bibliography

[1] The VANET/G5-C-ITS, Intelligent Transport Systems–Connected Vehicles Project at Middlesex University. Vehicular ad hoc network (vanet). online.

[2] Marco Annoni and Bob Williams. *The History of Vehicular Networks*, pages 3–21. Springer International Publishing, Cham, 2015.

[3] Seng W Loke. Cooperative automated vehicles: A review of opportunities and challenges in socially intelligent vehicles beyond networking. *IEEE Transactions on Intelligent Vehicles*, 4(4):509–518, 2019.

[4] Soodeh Dadras and Chris Winstead. Cybersecurity of autonomous vehicle platooning. presented in Utah State University, 2017.

[5] Walter Brenner and Andreas Herrmann. *An Overview of Technology, Benefits and Impact of Automated and Autonomous Driving on the Automotive Industry*, pages 427–442. Springer Berlin Heidelberg, 2018.

[6] Sean Campbell, Niall O'Mahony, Lenka Krpalcova, Daniel Riordan, Joseph Walsh, Aidan Murphy, and Conor Ryan. Sensor technology in autonomous vehicles : A review. In *2018 29th Irish Signals and Systems Conference (ISSC)*, pages 1–4, Belfast, United Kingdom, Jun. 21–22, 2018.

[7] Review of research on v2x technologies, strategies, and operations. *Renewable and Sustainable Energy Reviews*, 105:61–70, 2019.

[8] Hamid Khayyam, Bahman Javadi, Mahdi Jalili, and Reza N. Jazar. *Artificial Intelligence and Internet of Things for Autonomous Vehicles*, pages 39–68. Springer International Publishing, Cham, 2020.

[9] Yaqiong Liu, Mugen Peng, Guochu Shou, Yudong Chen, and Siyu Chen. Toward edge intelligence: Multiaccess edge computing for 5g and internet of things. *IEEE Internet of Things Journal*, 7(8):6722–6747, 2020.

[10] Lei Liu, Chen Chen, Qingqi Pei, Sabita Maharjan, and Yan Zhang. Vehicular Edge Computing and Networking: A Survey. *Mobile Networks and Applications*, 26(3):1145–1168, 2021.

[11] Xuefeng Xiao, Xueshi Hou, Xinlei Chen, Chenhao Liu, and Yong Li. Quantitative analysis for capabilities of vehicular fog computing. *Information Sciences*, 501:742–760, 2019.

[12] Sherin Abdelhamid, Hossam S. Hassanein, and Glen Takahara. Vehicle as a resource (vaar). *IEEE Network*, 29(1):12–17, 2015.

[13] Yifang Ma, Zhenyu Wang, Hong Yang, and Lin Yang. Artificial intelligence applications in the development of autonomous vehicles: a survey. *IEEE/CAA Journal of Automatica Sinica*, 7(2):315–329, 2020.

[14] Roy D. Yates, Yin Sun, D. Richard Brown, Sanjit K. Kaul, Eytan Modiano, and Sennur Ulukus. Age of information: An introduction and survey. *IEEE Journal on Selected Areas in Communications*, 39(5):1183–1210, 2021.

[15] Weisong Shi and Schahram Dustdar. The promise of edge computing. *Computer*, 49(5):78–81, 2016.

[16] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys Tutorials*, 19(3):1628–1656, 2017.

[17] Q. Hu, C. Wu, X. Zhao, X. Chen, Y. Ji, and T. Yoshinaga. Vehicular multi-access edge computing with licensed sub-6 ghz, ieee 802.11p and mmwave. *IEEE Access*, 6:1995–2004, 2018.

[18] Anthony D Josep, Randy Katz, Andy Konwinski, LEE Gunho, David Patterson, and Ariel Rabkin. A view of cloud computing. *Communications of the ACM*, 53(4), 2010.

[19] Abbas Kiani and Nirwan Ansari. Toward hierarchical mobile edge computing: An auction-based profit maximization approach. *IEEE Internet of Things Journal*, 4(6):2082–2091, 2017.

[20] Pouria Sayyad Khodashenas, Cristina Ruiz, J Ferrer Riera, Jose Oscar Fajardo, Ianire Taboada, Bego Blanco, Fidel Liberal, Javier Garcia Lloreda, Jordi Pérez-Romero, Oriol Sallent, et al. Service provisioning and pricing methods in a multi-tenant cloud enabled ran. In *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 1–6, Berlin, Germany, Oct. 31–Nov. 2, 2016. IEEE.

[21] K. Zhang, Y. Mao, S. Leng, Y. He, and Y. ZHANG. Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading. *IEEE Vehicular Technology Magazine*, 12(2):36–44, June 2017.

[22] Yang Zhang, Jing Zhao, and Guohong Cao. On scheduling vehicle-roadside data access. In *Proceedings of the Fourth ACM International Workshop on Vehicular Ad Hoc Networks*, VANET '07, pages 9–18, Sep. 10, 2007.

[23] K. Zhang, Y. Mao, S. Leng, S. Maharjan, and Y. Zhang. Optimal delay constrained offloading for vehicular edge computing networks. In *IEEE International Conference on Commun. (ICC)*, pages 1–6, Paris, France, May 21–25, 2017.

[24] Ke Zhang, Yuming Mao, Supeng Leng, Sabita Maharjan, Alexey Vinel, and Yan Zhang. Contract-theoretic approach for delay constrained offloading in vehicular edge computing networks. *Mobile Networks and Applications*, Feb 2018.

[25] K. Zhang, Y. Mao, S. Leng, S. Maharjan, and Y. Zhang. Optimal delay constrained offloading for vehicular edge computing networks. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6, Paris, France, May 21–25, 2017.

[26] J. Feng, Z. Liu, C. Wu, and Y. Ji. Ave: Autonomous vehicular edge computing framework with aco-based scheduling. *IEEE Transactions on Vehicular Technology*, 66(12):10660–10675, Dec 2017.

[27] R. Yu, J. Ding, X. Huang, M. T. Zhou, S. Gjessing, and Y. Zhang. Optimal resource sharing in 5g-enabled vehicular networks: A matrix game approach. *IEEE Transactions on Vehicular Technology*, 65(10):7844–7856, Oct 2016.

[28] Dennis Grewe, Marco Wagner, Mayutan Arumaithurai, Ioannis Psaras, and Dirk Kutscher. Information-centric mobile edge computing for connected vehicle environments: Challenges and research directions. In *Proceedings of the Workshop on Mobile Edge Communications*, pages 7–12, Los Angeles, CA, USA, Aug. 21 – 25, 2017.

[29] Ying He, Chengchao Liang, Zheng Zhang, F Richard Yu, Nan Zhao, Hongxi Yin, and Yanhua Zhang. Resource allocation in software-defined and information-centric vehicular networks with mobile edge computing. In *Vehicular Technology*

Conference (VTC-Fall), 2017 IEEE 86th, pages 1–5, Toronto, Ontario, Canada, Sep. 24–27, 2017.

[30] M. Deng, H. Tian, and X. Lyu. Adaptive sequential offloading game for multi-cell mobile edge computing. In *2016 23rd International Conference on Telecommunications (ICT)*, pages 1–5, Thessaloniki, Greece, May 16–18, 2016.

[31] X. Chen, L. Jiao, W. Li, and X. Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24(5):2795–2808, October 2016.

[32] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang. Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. *IEEE Access*, 4:5896–5907, 2016.

[33] Chenmeng Wang, Chengchao Liang, F Richard Yu, Qianbin Chen, and Lun Tang. Computation offloading and resource allocation in wireless cellular networks with mobile edge computing. *IEEE Transactions on Wireless Communications*, 16(8):4924–4938, 2017.

[34] Chenmeng Wang, F Richard Yu, Chengchao Liang, Qianbin Chen, and Lun Tang. Joint computation offloading and interference management in wireless cellular networks with mobile edge computing. *IEEE Transactions on Vehicular Technology*, 66(8):7432–7445, 2017.

[35] Xuyu Wang, Shiwen Mao, and Michelle X. Gong. An overview of 3gpp cellular vehicle-to-everything standards. *GetMobile: Mobile Comp. and Comm.*, 21(3):19–25, November 2017.

[36] A. Kousaridas, D. Medina, S. Ayaz, and C. Zhou. Recent advances in 3gpp networks for vehicular communications. In *IEEE Conference on Standards for*

Communications and Networking (CSCN), pages 91–97, Helsinki, Finland, Sep. 18–20, 2017.

[37] H. Zhou, W. Xu, Y. Bi, J. Chen, Q. Yu, and X. S. Shen. Toward 5g spectrum sharing for immersive-experience-driven vehicular communications. *IEEE Wireless Communications*, 24(6):30–37, Dec 2017.

[38] G. H. Sim, S. Klos, A. Asadi, A. Klein, and M. Hollick. An online context-aware machine learning algorithm for 5g mmwave vehicular communications. *IEEE/ACM Transactions on Networking*, 26(6):1–14, 2018.

[39] Andrea Tassi, Malcolm Egan, Robert J. Piechocki, and Andrew R. Nix. Modeling and design of millimeter-wave networks for highway vehicular communication. *IEEE Trans. on Vehicular Technology*, 66(12):10676–10691, 2017.

[40] B. Di, L. Song, Y. Li, and Z. Han. V2x meets noma: Non-orthogonal multiple access for 5g-enabled vehicular networks. *IEEE Wireless Communications*, 24(6):14–21, Dec 2017.

[41] Claudia Campolo, Antonella Molinaro, Antonio Iera, and Francesco Menichella. 5g network slicing for vehicle-to-everything services. *Wireless Commun.*, 24(6):38–45, December 2017.

[42] Tesnim Mekki, Issam Jabri, Lamia Chaari, and Abderrezak Rachedi. A survey on vehicular fog computing: Motivation, architectures, taxonomy, and issues. In Leonard Barolli, Flora Amato, Francesco Moscato, Tomoya Enokido, and Makoto Takizawa, editors, *Web, Artificial Intelligence and Network Applications*, pages 159–168, Cham, 2020. Springer International Publishing.

[43] Favian Dewanta and Masahiro Mambo. Bpt scheme: Establishing trusted vehicular fog computing service for rural area based on blockchain approach. *IEEE Transactions on Vehicular Technology*, 70(2):1752–1769, 2021.

[44] Junbin Liang, Jie Zhang, Victor C.M. Leung, and Xu Wu. Distributed information exchange with low latency for decision making in vehicular fog computing. *IEEE Internet of Things Journal*, pages 1–1, 2021. Early access.

[45] Salman Memon and Muthucumaru Maheswaran. Using machine learning for handover optimization in vehicular fog computing. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 182–190, Limassol, Cyprus, Apr. 8–12, 2019.

[46] Sheng Zhou, Yuxuan Sun, Zhiyuan Jiang, and Zhisheng Niu. Exploiting moving intelligence: Delay-optimized computation offloading in vehicular fog networks. *arXiv preprint arXiv:1902.09401*, 2019.

[47] Zhaolong Ning, Jun Huang, and Xiaojie Wang. Vehicular fog computing: Enabling real-time traffic management for smart cities. *IEEE Wireless Communications*, 26(1):87–93, 2019.

[48] Jéferson Campos Nobre, Allan M de Souza, Denis Rosario, Cristiano Both, Leandro A Villas, Eduardo Cerqueira, Torsten Braun, and Mario Gerla. Vehicular software-defined networking and fog computing: integration and design principles. *Ad Hoc Networks*, 82:172–181, 2019.

[49] Habtamu Mohammed Birhanie, Mohammed Ayoub Messous, Sidi-Mohammed

Senouci, El-Hassane Aglzim, and Ahmedin Mohammed Ahmed. Mdp-based resource allocation scheme towards a vehicular fog computing with energy constraints. In *IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, Abu Dhabi, UAE, Dec. 9–13, 2018.

[50] Zhenyu Zhou, Pengju Liu, Junhao Feng, Yan Zhang, Shahid Mumtaz, and Jonathan Rodriguez. Computation resource allocation and task assignment optimization in vehicular fog computing: A contract-matching approach. *IEEE Transactions on Vehicular Technology*, 68(4):3113–3125, 2019.

[51] Jindou Xie, Yunjian Jia, Zhengchuan Chen, and Liang Liang. Mobility-aware task parallel offloading for vehicle fog computing. In *International Conference on Artificial Intelligence for Communications and Networks*, pages 367–379, Harbin, China, May 25–26, 2019.

[52] Shi Yan, Xinran Zhang, Hongyu Xiang, and Wenbin Wu. Joint access mode selection and spectrum allocation for fog computing based vehicular networks. *IEEE Access*, 7:17725–17735, 2019.

[53] Zhenyu Zhou, Junhao Feng, Bo Gu, Bo Ai, Shahid Mumtaz, Jonathan Rodriguez, and Mohsen Guizani. When mobile crowd sensing meets uav: Energy-efficient task assignment and route planning. *IEEE Transactions on Communications*, 66(11):5526–5538, 2018.

[54] Peng Duan, Yunjian Jia, Liang Liang, Jonathan Rodriguez, Kazi Mohammed Saidul Huq, and Guojun Li. Space-reserved cooperative caching in 5g heterogeneous networks for industrial iot. *IEEE Transactions on Industrial Informatics*, 14(6):2715–2724, 2018.

[55] Xiaojie Wang, Zhaolong Ning, and Lei Wang. Offloading in internet of vehicles: A fog-enabled real-time traffic management system. *IEEE Transactions on Industrial Informatics*, 14(10):4568–4578, 2018.

[56] Jun Wu, Mianxiong Dong, Kaoru Ota, Jianhua Li, Wu Yang, and Meng Wang. Fog-computing-enabled cognitive network function virtualization for an information-centric future internet. *IEEE Communications Magazine*, 57(7):48–54, 2019.

[57] Xianfu Chen and et.al. Age of information aware radio resource management in vehicular networks: A proactive deep reinforcement learning perspective. *IEEE Transactions on Wireless Communications*, 19(4):2268 – 2281, 2020.

[58] Mohamed K Abdel-Aziz, Sumudu Samarakoon, Chen-Feng Liu, Mehdi Bennis, and Walid Saad. Optimized age of information tail for ultra-reliable low-latency communications in vehicular networks. *IEEE Transactions on Communications*, 68(3):1911 – 1924, 2019.

[59] Mohamed K Abdel-Aziz, Sumudu Samarakoon, Mehdi Bennis, and Walid Saad. Ultra-reliable and low-latency vehicular communication: An active learning approach. *IEEE Communications Letters*, 24(2):367 – 370, 2019.

[60] Mohamed K Abdel-Aziz, Chen-Feng Liu, Sumudu Samarakoon, Mehdi Bennis, and Walid Saad. Ultra-reliable low-latency vehicular networks: Taming the age of information tail. In *IEEE Global Communications Conference (GLOBECOM)*, pages 1–7, Abu Dhabi, UAE, Dec. 9–13, 2018.

[61] Sanjit Kaul, Marco Gruteser, Vinuth Rai, and John Kenney. Minimizing age of information in vehicular networks. In *8th Annual IEEE Communications Society*

Conference on Sensor, Mesh and Ad Hoc Communications and Networks, pages 350–358, Salt Lake City, Utah, USA, Jun. 27–30, 2011.

[62] Yuanzhi Ni, Lin Cai, and Yuming Bo. Vehicular beacon broadcast scheduling based on age of information (AoI). *China Communications*, 15(7):67–76, 2018.

[63] Maohong Chen, Yong Xiao, Qiang Li, and Kwang-cheng Chen. Minimizing age-of-information for fog computing-supported vehicular networks with deep q-learning. *arXiv preprint arXiv:2004.04640*, 2020.

[64] Ning Lu, Bo Ji, and Bin Li. Age-based scheduling: Improving data freshness for wireless real-time traffic. In *Proceedings of the eighteenth acm international symposium on mobile ad hoc networking and computing*, pages 191–200, Los Angeles, CA, USA, Jun. 26–29, 2018.

[65] Igor Kadota and Eytan Modiano. Minimizing the age of information in wireless networks with stochastic arrivals. *IEEE Transactions on Mobile Computing*, 20(3):1173–1185, 2021.

[66] Nikolaos Pappas, Johan Gunnarsson, Ludvig Kratz, Marios Kountouris, and Vangelis Angelakis. Age of information of multiple sources with queue management. In *2015 IEEE International Conference on Communications (ICC)*, pages 5935–5940, London, UK, Jun. 8–12, 2015.

[67] Abubakr Alabbasi and Vaneet Aggarwal. Joint information freshness and completion time optimization for vehicular networks. *IEEE Transactions on Services Computing*, in press.

[68] Ignacio Llatser, Andreas Festag, and Gerhard Fettweis. Vehicular communication performance in convoys of automated vehicles. In *IEEE International Conference*

on Communications (ICC), pages 1–6, Kuala Lumpur, Malaysia, May 23–37, 2016.

[69] Evangelos Mintsis, Leonhard Lücken, Vasilios Karagounis, Kallirroi Porfyri, Michele Rondinone, Alejandro Correa, Julian Schindler, and Evangelos Mitsakis. Joint deployment of infrastructure-assisted traffic management and cooperative driving around work zones. In *IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8, Sep. 20–23, 2020. virtual conference.

[70] Gokulnath Thandavarayan, Miguel Sepulcre, and Javier Gozalvez. Generation of cooperative perception messages for connected and automated vehicles. *IEEE Transactions on Vehicular Technology*, 69(12):16336–16341, 2020.

[71] Baldomero Coll-Perales, Joschua Schulte-Tigges, Michele Rondinone, Javier Gozalvez, Michael Reke, Dominik Matheis, and Thomas Walter. Prototyping and evaluation of infrastructure-assisted transition of control for cooperative automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–17, 2021. Early access.

[72] Ibrahim Sorkhoh, Dariush Ebrahimi, Ribal Atallah, and Chadi Assi. Workload scheduling in vehicular networks with edge cloud capabilities. *IEEE Transactions on Vehicular Technology*, 68(9):8472–8486, 2019.

[73] Michael L Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2016.

[74] Ibrahim Sorkhoh, Dariush Ebrahimi, Chadi Assi, Sanaa Sharafeddine, and Maurice Khabbaz. An infrastructure-assisted workload scheduling for computational resources exploitation in the fog-enabled vehicular network. *IEEE Internet of Things Journal*, 7(6):5021–5032, 2020.

[75] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B. Letaief. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys Tutorials*, 19(4):2322–2358, 2017.

[76] Elie El Haber, Tri Minh Nguyen, and Chadi Assi. Joint optimization of computational cost and devices energy for task offloading in multi-tier edge-clouds. *IEEE Transactions on Communications*, 67(5):3407–3421, 2019.

[77] Peter Brucker. *Scheduling Algorithms*. SpringerVerlag, 2004.

[78] L.A. Wolsey. *Integer Programming*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1998.

[79] Ibrahim Sorkhoh, Chadi Assi, Dariush Ebrahimi, and Sanaa Sharafeddine. Optimizing information freshness for mec-enabled cooperative autonomous driving. *IEEE Transaction on Intelligent Transportation Systems*, 2021 (SUBMITTED).

[80] Yin Sun, Yury Polyanskiy, and Elif Uysal-Biyikoglu. Remote estimation of the wiener process over a channel with random delay. In *IEEE International Symposium on Information Theory (ISIT)*, pages 321–325, Aachen, Germany, Jun. 25–30, 2017.

[81] Bo Zhou, Walid Saad, Mehdi Bennis, and Petar Popovski. Risk-aware optimization of age of information in the internet of things. *arXiv preprint arXiv:2002.09805*, 2020.

[82] Svetlana A Kravchenko and Frank Werner. Erratum to: Minimizing total tardiness on parallel machines with preemptions. *Journal of Scheduling*, 16(4):439–441, 2013.

[83] Svetlana A Kravchenko and Frank Werner. Minimizing total tardiness on parallel machines with preemptions. *Journal of Scheduling*, 15(2):193–200, 2012.

[84] Vasek Chvatal, Vaclav Chvatal, et al. *Linear programming*. Macmillan, 1983.

[85] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*, Maui, Hawaii, USA, Nov. 4–7, 2018.

[86] German Aerospace Center. Sumo documentation. Accessed: 2021-09-11.

[87] IBM ILOG Cplex. V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.

[88] Ibm. `https://www.ibm.com/ca-en`. Accessed: 2021-09-11.

Chapter 7

Appendix

Appendix A

Optimization Techniques

This chapter briefly explains the optimization techniques we apply to solve various problems we encounter in this thesis. Notice that the methods described here are only integer linear programming techniques. We will not discuss methods applicable for other kinds of problems (e.g., convex problems).

To discuss these techniques, we will refer to the following general integer linear program:

$$\begin{aligned} & \min_{\mathbf{x}} \mathbf{c}.\mathbf{x} \\ s.t. & \\ & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{D}\mathbf{x} \leq \mathbf{e} \\ & \mathbf{x} \in \mathbf{X} \end{aligned} \tag{A.1}$$

$\mathbf{A}$ and $\mathbf{D}$ are matrices. $\mathbf{x}$ is the variable vector. $\mathbf{c}$, $\mathbf{b}$ and $\mathbf{e}$ are all constant vectors.

A.1 Lagrangian Relaxation (LR)

LR is a technique that utilizes the Lagrangian duality theory to simplify the program via relaxing some constraints. After careful studies of the problem ILP, we might realize that removing some constraints will lead to a much easier problem (i.e., polynomial-time solvable). However, LR does not entirely ignore the relaxed constraints. It penalizes the violation of these constraints through their inclusion in the program objective function multiplied by what is called a dual variable. An LR for A.1 can be as follows:

$$\begin{aligned} & \min_{\mathbf{x}} \mathbf{c}.\mathbf{x} + \boldsymbol{\lambda}\big(e - \mathbf{D}\mathbf{x}\big) \\ s.t. & \\ & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & x \in \mathbf{X} \end{aligned} \tag{A.2}$$

As shown in A.2, to apply an LR for a program, we subtract the left-hand side of the constraint from its right-hand side. Then we multiply the result with the corresponding dual variable. Most probably, such a problem is easier to solve than the original problem. Finding the proper values for the dual variables is the critical part of any LR approach. The following is called the Lagrangian Dual problem:

$$\begin{aligned} \max_{\boldsymbol{\lambda}} \big(\quad & \min_{\mathbf{x}} \mathbf{c}.\mathbf{x} + \boldsymbol{\lambda}\big(e - \mathbf{D}\mathbf{x}\big) \\ & s.t. \\ & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbf{X} \ \big) \end{aligned} \tag{A.3}$$

For minimization problem, the dual program tries to find a good lower bound. To do so, it tries to maximize the objective reached by the dual program so it becomes

closer to the original problem optimum solution. The challenge is that we have an exponential number of constraints in the dual problem. To tackle this challenge, LR usually applies the Subgradient Descent Method (SBGD). It is a variation of the original gradient descent that is applicable with functions that are not continuous. The following is the equation to update the dual variable in each iteration:

$$\boldsymbol{\lambda}^{\tau+1} = \boldsymbol{\lambda}^{\tau} + \frac{\epsilon * [INFS(\boldsymbol{\lambda}^{\tau}) - FS(\boldsymbol{\lambda}^{\tau})]}{\|S\|^2} S \tag{A.4}$$

where S is the subtraction of the relaxed constraints' left-hand side from the right-hand side. ϵ is a small number and tau is the iteration counter. FS stands for feasible solution and $INFS$ stands for infeasible one. The last part of the LR methods is constructing a feasible solution from the infeasible solution created by solving A.2. This part is a problem by itself and should be considered while choosing the constraint to relax. The LR method will produce a better objective if this part is solvable to optimality in acceptable time.

A.1.1 Integrality property

Let P be the original problem. Let $\dot{P}$ be the linear relaxation to the problem. Let LR(P) be the Lagrangian relaxation to problem P. Then consider this: If LR(P) solution set is the convex hull of the relaxed problem, then $OPT(LR(P)) = OPT(LR(\dot{P}))$. And since $OPT(\dot{P}) = OPT(LR(\dot{P}))$ (duality theory), then $OPT(\dot{P}) = OPT(LR(P))$. It means that the LR you chose for the problem is as bad as the linear relaxation of the problem. Simply, it means that you relaxed the problem too much. A method designer should be aware of such property and examine whether the resulted model is the convex hull of the relaxed problem or not. A model is the convex hull of the problem if its constraints matrix is unimodular.

A.2 Dantzing-Wolfe Decomposition (DW)

Consider that A.1 can be written as follows:

$$\min_{\mathbf{x}} \mathbf{c}.\mathbf{x} \tag{A.5}$$

s.t.

$$\begin{array}{llllll}
\mathbf{A}^1\mathbf{x}^1 & +\mathbf{A}^2\mathbf{x}^2 & ... & +\mathbf{A}^K\mathbf{x}^K & \leq & \mathbf{b} \\
\mathbf{D}^1\mathbf{x}^1 & & & & \leq & \mathbf{d_1} \\
 & ... & & & \leq & . \\
 & & ... & & \leq & . \\
 & & & \mathbf{D}^K\mathbf{x}^K & \leq & \mathbf{d_K} \\
\mathbf{x}^1 \in Z_+^{n1} & ... & ... & \mathbf{x}^K \in Z_+^{nK} & &
\end{array}$$

This form is called the blocks structure form. If we can separate the model into several smaller ones, each for one block, we will probably end with sub-problems that are easier to solve. Here where the DW decomposition plays its role. DW is a decomposition technique based on the presentation theory of linear programming, which states the following; Any feasible solution in a problem closed set can be written as a affine combination of the extreme points of the set. DW takes this rule and decomposes the problem by rewriting the program in a way that shows any solution as a bunch of affine combinations of solutions coming from several smaller sub-problems. The DW model of A.1 is the following:

$$
\begin{aligned}
&\min_{\mathbf{x}} \sum_{k \le K} \sum_{j \in J_k} \delta_j^k \mathbf{c}^k . \mathbf{x}_j^k \\
&s.t. \\
&\sum_{k \le K} \sum_{j \in J_k} \delta_j^k \mathbf{A}^k \mathbf{x}_j^k \le \mathbf{b} \\
&\sum_{j \in J_k} \delta_k = 1 \qquad\qquad \forall k \le K \\
&\delta_j^k \in \{0,1\} \qquad\qquad \forall k \le K \;\; \forall j \in J_k \\
&\mathbf{x}_j^k \in \mathbf{X_k} \qquad\qquad \forall k \le K
\end{aligned}
\tag{A.6}
$$

The number of extreme points in a typical linear program set is exponential. Hence the number of variables in A.6 is enormous. To overcome this challenge, the program model tableau is written in the revised simplex method form; Only the basis variables (columns) are present in the tableau. The method adds the new variable by calculating their reduced cost coefficients by utilizing the constraints dual variables. The dual variable can be determined directly from the basis columns. This is in the typical revised simplex method scenario where the number of variables is tolerable. When we need to avoid the direct calculation of every non-basis variables reduced costs, we can utilize the columns generation method (CG). In CG, we should formulate another optimization problem where its constraints are the blocks constraints shown in A.5, and its objective function is the reduced cost formula:

$$
\max_{\mathbf{x}^k} \mathrm{RCC}(\boldsymbol{x}^k) = (\boldsymbol{\omega}^k \mathbf{A}^k - \boldsymbol{c}^k)\boldsymbol{x}^k + \gamma^k \tag{A.7}
$$

$\boldsymbol{\omega}^k$ is the dual variable vector of the contraints in D^k, and γ^k is the dual variable of the the second constraint in A.6. In each iteration of the simplex method, CG solves these subproblems and generates columns to add to the basis, the master tableau. Then, the master calculates the dual variables ($\boldsymbol{\omega}^k$) and sends it again the subproblems. It

keeps iterating until a certain criterion is met. Then, it applies the final step, which is turning the master variables back into integers and solving the ILP generated.

A.3 Benders Decomposition (BS)

Compared to the previously discussed techniques, BS is the most advanced and, most of the time, efficient decomposition technique. That's because it can generate a feasible solution in each iteration, unlike DW. At the same time, it can be applied to any ILP regardless of whether it is easy to construct a feasible solution or not, unlike LR. The only drawback is the complexity of solving the master problem (an integer program) and the cut generation.

BS starts by deciding the variables that belong to the master problem and those that belong to the subproblem/s. The choice should be based on the simplicity of the generated subproblem and the BS cut. Notice that the subproblem should be solved to optimality. Otherwise, the cut will remove some feasible solutions from the set. Once chosen, the designer should provide an efficient and optimum solution to the subproblems and a valid cut. A cut is a message that the subproblems send to the master to inform him how good it did in the previous iteration.

Consider the following ILP:

$$\begin{aligned} & \min_{\mathbf{x},\mathbf{y}} \mathbf{c}.\mathbf{x} + \mathbf{h}.\mathbf{y} \\ s.t. & \\ & \mathbf{A}\mathbf{x} + \mathbf{D}\mathbf{y} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbf{X} \\ & \mathbf{y} \in \mathbf{Y} \end{aligned} \tag{A.8}$$

$\mathbf{y}$ is a variable vector. A BS master problem for such an ILP can be as follows:

$$\begin{aligned} &\min_{\mathbf{x}} \mathbf{c}.\mathbf{x} + \mathbf{h}.\bar{\mathbf{y}} \\ s.t. \\ &\mathbf{A}\mathbf{x} + \mathbf{D}\bar{\mathbf{y}} \leq \mathbf{b} \\ &\mathbf{x} \in \mathbf{X} \end{aligned} \tag{A.9}$$

$\bar{\mathbf{y}}$ is a vector variable assignment. When the master problem is solved, it fixes its variables $\mathbf{x}$ to certain values. These values are sent as constants to the subproblems that should find the best solution to its variables $\mathbf{y}$, while considering these variables constraints, and generates a cut of the following form:

$$z \geq \zeta_k \times \left(\bigwedge_{o \in O} I(\mathbf{v}_k^o \mathbf{x} \leq r_k) \right) \tag{A.10}$$

z is equal to the objective function, k is the iteration, ζ_k is the objective value reached in the the k^{th} iteration, $\mathbf{v}_k^o$ is a vector of constants, and r_k is a constant. The cut states that if the conditions in the indications functions are all true, then the objective will be grater than or equal to ζ_k This cut should be added to the master problem as a constraint. Then the master should update its variables' values and send them again to the subproblems. This procedure will keep repeating until the convergence to the optimal solution or other conditions (e.g., number of iterations).

Appendix B

Software Tools

B.1 Simulation of Urban MObility (SUMO)

SUMO [85] is a portable, microscopic simulator for urban traffic designed by the German Aerospace Center [86]. It has been open-source and available since 2001. Its purpose is to test traffic strategies, and it is currently used for traffic forecasting, evaluation of traffic lights, navigation systems research, and many other applications.

SUMO is a time-based simulator. In every time unit, it generates an arrival of a vehicle to the constructed road network via a geometric probability distribution. This probability is called *the emitting probability*. This is the case when we set the arrival process to random. It is possible to predefine the arrival of the vehicles at specific times. Before the simulation, the experimenter must construct the road network and specify various parameters such as the vehicle's average speed, the cars' length, and the arrival rate.

In this thesis, we use SUMO to evaluate our proposed solutions. There are two modes that we considered in our evaluation:

1. **Offline mode**: We generate a particular instance of the problem we are trying to solve. This instance is to evaluate the method only once. We run the simulator for sufficient time to make the system passes its transient response time. Once this occurs, we construct an instance of the problem by randomly selecting the vehicles as part of the problem instance.

2. **Online mode**: We try to evaluate the overall system performance by accumulating the solution of the scheduling problem over the resources of the systems (e.g., the computational resource and the wireless channels). We do that by collecting a longitudinal data stream from sumo that specifies the properties of the cars (location, speed, etc.) in each time unit. Then, using this data, we keep constructing instances of the problem we are trying to solve and assign the load to the available resources. This long-term evaluation allows us to assess the system's performance by changing various parameters like the resources capacity and the load arrival.

B.2 IBM ILOG CPLEX Optimization Studio

CPLEX [87] is an optimization software package that solves linear and linear-integer problems. It is named after the simplex method. The software provides C++, JAVA, and Python APIs to model the problem and apply off-the-shelf algorithms. For linear programs, CPLEX can use the simplex method or the Barrier interior-point method. It can perform several branch-and-bound techniques for integer-linear programs, but its implemented branch-and-cut method is the most known. It is a product of IBM [88].

In this thesis, we assess the performance of the proposed algorithm by comparing the time and objective with the CPLEX outcome. We strive to model each problem

as ILP and feed it to CPLEX to find the time and the optimal solution objective. Then we run our solution and compare both methods.