

AUS Repository

An Auction-Based Scheduling Approach for Minimizing Latency in Fog Computing Using 5G Infrastructure

Item Type	Thesis
Authors	Fahmy, Ahmed
Download date	2026-04-16 12:09:44
Link to Item	http://hdl.handle.net/11073/16647

AN AUCTION-BASED SCHEDULING APPROACH FOR MINIMIZING
LATENCY IN FOG COMPUTING USING 5G INFRASTRUCTURE

by

Ahmed Fahmy

A Thesis Presented to the Faculty of the
American University of Sharjah
College of Engineering
In Partial Fulfillment
of the Requirements
for the Degree of

Master of Science in
Computer Engineering

Sharjah, United Arab Emirates

February 2020

Declaration of Authorship

I declare that this thesis is my own work and, to the best of my knowledge and belief, it does not contain material published or written by a third party, except where permission has been obtained and/or appropriately cited through full and accurate referencing.

Signature: *Ahmed Fahmy*

Date: 03/03/2020

The Author controls copyright for this report.
Material should not be reused without the consent of the author. Due
acknowledgement should be made where appropriate.

© Year 2020

Ahmed Fahmy

ALL RIGHTS RESERVED

Approval Signatures

We, the undersigned, approve the Master's Thesis of Ahmed Fahmy

Thesis Title: An Auction-Based Scheduling Approach for Minimizing Latency in Fog Computing Using 5G Infrastructure.

Signature

Date of Signature
(dd/mm/yyyy)

Dr. Raafat Aburukba
Assistant Professor, Department of Computer Science and Engineering
Thesis Advisor

Dr. Taha Landolsi
Professor, Department of Computer Science and Engineering
Thesis Co-Advisor

Dr. Khaled El Fakih
Professor, Department of Computer Science and Engineering
Thesis Committee Member

Dr. Mahmoud H. Ismail
Associate Professor, Department of Electrical Engineering
Thesis Committee Member

Dr. Fadi Aloul
Head, Department of Computer Science and Engineering

Dr. Lotfi Romdhane
Associate Dean for Graduate Affairs and Research
College of Engineering

Dr. Sirin Tekinay
Dean, College of Engineering

Dr. Mohamed El-Tarhuni
Vice Provost for Graduate Studies

Acknowledgment

I would like to thank my family for everything they have done and for the sacrifices and love they shared to help me reach this new stage in my life. I am very grateful to you for always providing me with all the best life has to offer and pushing me beyond my limits to achieve new heights. I would also like to thank my friends for their consistent love and support. Thank you for being there when I needed you and for believing in me. I love you all very much.

I would like to thank my advisors Dr. Raafat Aburukba and Dr. Taha Landolsi for providing knowledge, guidance, support, and motivation throughout my research stages. I am deeply beholden for their great assistance, worthy discussions, and suggestions. It was a great pleasure working with you on this research.

Finally, I would like to thank the American University of Sharjah and the Department of Computer Science and Engineering for granting me a graduate assistantship during my study.

Dedication

To my niece, Lulu...

Abstract

The advent of the Internet of Things (IoT) has brought an unprecedented increase in the number of connected devices. Recently, IoT-based devices have been used in several applications including healthcare, data analytics, smart cities, and many others. Time-sensitive applications, such as Vehicle-to-Vehicle (V2V) communication, led to the need for an Ultra-High Reliable Low Latency Communication (URLLC). Consequently, 5G networks gained massive attention from the research community due to its ability to support enormous amounts of transfer rate. One of the main supporting computing paradigms for IoT is cloud computing, as it offers computing capabilities over the Internet. Nevertheless, cloud computing is unsuitable for time-critical applications. Hence, researchers proposed deploying fog computing as part of the 5G small cells to tackle the deficiencies of cloud computing. Many challenges arise while combining 5G technology and fog computing such as scheduling service requests across small cells to reduce the overall latency. In this work, the scheduling problem is modeled as an optimization problem with the objective of minimizing the overall latency. Furthermore, small cells are decentralized by nature. Therefore, a coordination framework is proposed to handle the interdependency between small cells. Accordingly, the decentralized scheduling problem is mapped to a combinatorial auction optimization problem. The proposed optimization model is validated using an optimization engine. The scheduling problem is known as an NP-hard problem. Thus, a decentralized heuristic solution is proposed to solve the scheduling problem in polynomial time. The proposed solution integrates a novel Simulated Annealing-Based Scheduling (SABS) and Auction-Based Winner Determination (ABWD) heuristic algorithms. To assess the performance and quality of the proposed heuristic solution, a centralized approach is used as a benchmark. Furthermore, sensitivity analysis is conducted in which the impact of each system parameter on the system behavior is investigated. The results prove the adequacy of the proposed solution as the execution time remained approximately constant, with an average of 726 μ s, considering different problem sizes. Moreover, the proposed solution is found to be scalable and accommodates the exponential growth of IoT devices.

Keywords: *Fog Computing; IoT; 5G networks; Agent; Auction; Scheduling; Optimization; Game Theory; Decentralized.*

Table of Contents

Abstract	6
List of Figures	11
List of Tables.....	13
List of Algorithms	14
List of Abbreviations	15
Chapter 1. Introduction.....	16
1.1. Fog Computing.....	16
1.1.1. Edge tier	16
1.1.2. Fog tier	17
1.1.3. Cloud tier.....	17
1.2. 5G Network.....	18
1.2.1. C-RAN	18
1.2.2. Small cells	18
1.3. Scheduling Problem	19
1.4. Thesis Objectives	19
1.5. Research Contribution	20
Chapter 2. Background and Literature Review.....	21
2.1. Fog Computing Challenges	21
2.1.1. Privacy and security.....	21
2.1.2. Mobility.....	21
2.1.3. Interdependency.....	21
2.1.4. Limited resources.....	22
2.2. Suitability of Fog Computing	22
2.3. 5G Requirements and Characteristics	22
2.3.1. Enhanced mobile broadband	22
2.3.2. Ultra-reliable and low latency communication	22
2.3.3. Massive machine-type communication.....	23
2.4. Task Scheduling.....	23
2.4.1. Centralized Scheduling	24
2.4.2. Decentralized Scheduling.....	24
2.4.3. Complexity	24
2.4.4. Common Performance Metrics.....	25
2.5. Mechanism Design and Game Theory	26

2.5.1. Non-cooperative vs. Cooperative Games.....	27
2.5.2. Sequential vs. Simultaneous Games	27
2.5.3. Information in Games	28
2.5.4. Combinatorial Games	28
2.6. Related Work	28
Chapter 3. Scheduling Problem Modeling.....	35
3.1. Environment Overview.....	35
3.2. Modeling Resources	35
3.3. Modeling Time Dimension.....	36
3.4. Modeling Service Request.....	36
3.5. Modeling Dependency.....	38
3.6. Modeling Latency in 5G Environment.....	39
3.5.1. Propagation delay	39
3.5.2. Transmission delay	40
3.5.3. Compute delay.....	40
3.7. Problem Formulation.....	41
3.8. Model Validation.....	43
Chapter 4. Decentralized Approach	48
4.1. Environment Overview.....	48
4.2. Proposed Decentralized Framework	51
4.3. Self-Classification	54
4.3.1. Problem Formulation	54
4.3.2. Model Validation	56
4.4. Local Scheduling.....	59
4.4.1. Problem Formulation	60
4.4.2. Model Validation	60
4.5. Winner Determination	62
4.5.1. Problem Formulation	62
4.5.2. Agent preferences.	64
4.5.3. Utility calculation and bid generation.....	67
4.5.4. Model Validation	68
Chapter 5. Decentralized Auction-Based Heuristic Solution.....	72
5.1 Decentralized Auction-Based Solution Overview	72
5.2 Simulated Annealing-Based Scheduling (SABS) Algorithm	75

5.2.1.	SABS solution representation.....	77
5.2.2.	Initializing a starting solution.....	78
5.2.3.	Getting the objective value.....	79
5.2.4.	Finding the next solution.....	79
5.2.5.	The sequence function	81
5.2.6.	The “isFit” function	81
5.2.7.	Checking the dependency constraint	82
5.2.8.	SABS algorithm: complexity analysis	83
5.2.9.	SABS algorithm: example.....	83
5.3	Auction-Based Winner Determination (ABWD) Algorithm.....	88
5.3.1.	Preferences approximation.....	88
5.3.2.	Winner determination and solution representation.....	89
5.3.3.	Winner determination algorithm	90
5.3.4.	Utility calculation	92
5.3.5.	ABWD algorithm: example.....	93
Chapter 6.	Experimentation and Results.....	95
6.1.	Data Generation.....	95
6.2.	Centralized Approach Experimentations.....	95
6.2.1.	CPLEX simulation results.....	96
6.2.2.	SABS algorithm performance evaluation	98
6.2.3.	SABS sensitivity analysis	101
6.3.	Decentralized Approach Experimentations	110
6.3.1.	CPLEX simulation results.....	110
6.3.2.	Decentralized auction-based approach performance evaluation	112
6.3.3.	Decentralized auction-based heuristic sensitivity analysis	116
Chapter 7.	Case Study: Chronic Obstructive Pulmonary Disease	120
7.1.	Fog Computing in Healthcare.....	120
7.2.	Breath Assistance System.....	121
7.2.1.	Oxygen adjustment as a service request	122
7.2.2.	Simulation Results	122
Chapter 8.	Conclusion and Future Work	124
References	127
Appendix A – Scheduling Problem Model Validation.....		131
Appendix B – Self-Classification and Local Scheduling Model Validation.....		140

Appendix C – Winner Determination Model Validation	144
Vita	146

List of Figures

Figure 1.1: Fog computing architecture	17
Figure 2.1: 5G requirements and characteristics	23
Figure 3.1: Fog computing/5G architecture	36
Figure 3.2: The workflow of an example request	37
Figure 3.3: Gantt chart shows example allocation 1	45
Figure 3.4: Gantt chart shows example allocation 2	46
Figure 3.5: Gantt chart shows an optimal allocation of all operations by CPLEX	47
Figure 4.1: The interaction structure between agents	53
Figure 4.2: The value of d_j before and after the deadline	55
Figure 4.3: Gantt chart shows a potential solution of self-classification for agent $i = 1$	57
Figure 4.4: Gantt chart shows a potential solution of self-classification for agent $i = 2$	59
Figure 4.5: Gantt chart shows an example of local scheduling for agent $i = 1$	61
Figure 4.6: Gantt chart shows CPLEX local scheduling for agent $i = 2$	62
Figure 4.7: All combinations of available resource blocks (example)	66
Figure 4.8: Gantt chart shows CPLEX local scheduling for agent $i = 3$	69
Figure 4.9: Gantt chart shows the allocation of handed-over request $j = 1$ in agent $i = 1$	70
Figure 4.10: Gantt chart shows the allocation of handed-over request $j = 1$ in agent $i = 3$	71
Figure 5.1: A diagram shows the proposed solution workflow	74
Figure 5.2: SABS solution representation	77
Figure 5.3: Allocation representation (example)	77
Figure 5.4: Allocation for time slot $t = 0$ (starting solution)	84
Figure 5.5: Allocation for time slot $t = 1$ (starting solution)	85
Figure 5.6: Allocation for time slot $t = 2$ (starting solution)	85
Figure 5.7: Allocation for time slot $t = 3$ (starting solution)	86
Figure 5.8: The initial allocation for SABS algorithm	86
Figure 5.9: The allocation after finding the next solution	87
Figure 5.10: The available resources encoded by a provider agent (shaded area)	89
Figure 5.11: Inputs/outputs of the winner determination algorithm	90
Figure 6.1: Centralized approach execution time (CPLEX)	97
Figure 6.2: Centralized approach average makespan (CPLEX)	97
Figure 6.3: SABS vs. CPLEX (average makespan)	99
Figure 6.4: SABS average makespan	100
Figure 6.5: SABS execution time	101
Figure 6.6: SABS average makespan by varying the number of requests	102
Figure 6.7: SABS execution time by varying the number of requests	103
Figure 6.8: SABS average makespan vs. number of operations per request	104
Figure 6.9: SABS execution time vs. number of operations per request	105
Figure 6.10: SABS average makespan vs. the number of small cells	106
Figure 6.11: SABS execution time with different numbers of small cells	107

Figure 6.12: SABS average makespan vs. the number of compute blocks	108
Figure 6.13: SABS execution time vs. the number of compute blocks	109
Figure 6.14: Decentralized approach execution time (CPLEX)	111
Figure 6.15: Decentralized approach average makespan (CPLEX)	112
Figure 6.16: Decentralized heuristic vs. CPLEX (average makespan)	113
Figure 6.17: Centralized heuristic vs decentralized heuristic (average makespan) ...	115
Figure 6.18: Centralized heuristic vs. decentralized heuristic (execution time)	115
Figure 6.19: Average makespan by varying the number of provider agents	117
Figure 6.20: Average makespan by varying the number of consumer agents	118
Figure 6.21: Execution time vs. number of agents	118
Figure 7.1: Fog/5G environment of the breath assistance system	121

List of Tables

Table 3.1: Truth table shows dependency relationship	38
Table 3.2: Summary of all notations	41
Table 3.3: Summary of system parameter values	44
Table 3.4: Requests information	44
Table 3.5: Dependency information 1	45
Table 4.1: Summary of the system parameter values for the decentralized models ...	56
Table 4.2: Agents and requests information 1	57
Table 4.3: Dependency information 2	58
Table 4.4: Agents and requests information 2	69
Table 4.5: Dependency information of agent 3	69
Table 5.1: All operations before sorting	83
Table 5.2: All operations after sorting	84
Table 5.3: List of resources	94
Table 5.4: List of handed-over requests	94
Table 6.1: CPLEX experiments for the centralized approach	96
Table 6.2: CPLEX results for the centralized approach	96
Table 6.3: SABS results for the centralized approach	98
Table 6.4: SABS experimentation parameters and results	99
Table 6.5: The impact of varying the number of requests on SABS algorithm	102
Table 6.6: The impact of varying the number of operations on SABS algorithm	104
Table 6.7: The impact of varying the number of small cells on SABS algorithm	106
Table 6.8: The impact of varying the number of compute blocks on SABS algorithm	108
Table 6.9: CPLEX experiments for the decentralized approach	111
Table 6.10: CPLEX results for the decentralized approach	111
Table 6.11: The decentralized heuristic average makespan and execution time	113
Table 6.12: Decentralized heuristic experimentation parameters	114
Table 6.13: Experimental results: centralized vs. decentralized approach	114
Table 6.14: The impact of the number of provider agents on ABWD algorithm	116
Table 6.15: The impact of the number of consumer agents on ABWD algorithm	117
Table 7.1: Operations of a service request (oxygen adjustment)	122

List of Algorithms

Algorithm 5.1: Simulated annealing-based scheduling algorithm	76
Algorithm 5.2: Algorithm for initializing a starting solution	78
Algorithm 5.3: Algorithm for getting the objective value	79
Algorithm 5.4: Finding the next solution	80
Algorithm 5.5: The sequence function	81
Algorithm 5.6: The “isFit” function	82
Algorithm 5.7: Dependency constraint check	82
Algorithm 5.8: Winner determination	91
Algorithm 5.9: Utility calculation	92

List of Abbreviations

ABWD	Auction-Based Winner Determination
BBU	Base Band Units
C-RAN	Cloud-RAN
COPD	Chronic Obstructive Pulmonary Disease
CP	Constraint Programming
CPRI	Common Public Radio Interface
CS-LAT	Computational Size – Latency
DCN	Data Centre Networks
eMBB	Enhanced Mobile Broadband
IoT	Internet of Things
ISP	Internet Service Provider
LCG	Linked Conflict Graph
LoRaWAN	Long Range Wide Area Network
mMTC	Massive Machine Type Communication
OFDMA	Orthogonal Frequency-Division Multiple Access
OPL	Optimization Programming Language
PCC	Patient-Centered Care
RAN	Radio Access Network
RRH	Remote Radio Heads
SABS	Simulated Annealing-Based Scheduling
SCM	Small Cells Manager
URLLC	Ultra-High Reliable Low Latency Communication
V2V	Vehicle-to-Vehicle

Chapter 1. Introduction

With the massive growth of internet-connected devices and the prosperous evolution of the IoT industry, networks are congested with continuous service requests. Furthermore, the adoption of advanced technologies enabled time-critical applications that require real-time response, such as self-driving vehicles, disaster detection and early warning systems, and on-time clinical intervention. 5G technology, the current generation of cellular networks, is characterized by Enhanced Mobile Broadband (eMBB), Ultra-Reliable and Low Latency Communication (URLLC) and Massive Machine-Type Commutation (mMTC). Therefore, it is considered to be a prospective solution [1]. Nevertheless, the cloud computing architecture is inadequate to provide the required level of low latency communication. Hence, Cisco coined a new solution called fog computing by which the capabilities of the cloud paradigm is extended nearer to the edge [2].

1.1. Fog Computing

Fog computing is an extending scheme to the cloud paradigm by which the cloud computing services are provided nearer to the edge. The implementation of fog computing can be anywhere in the network between the data source and the cloud data centers such that the physical distance between the service provider and the service request generating devices is reduced. Consequentially, the delay in time required to respond to the service request is reduced to meet the requirements of real-time applications.

As fog computing is still an emerging technology, various deployments are considered in the literature. For example, the researchers in [3] considered mobile user-devices as fog units. On the other hand, fog computing is implemented as part of the network in [4], i.e. routers, switches or gateways. In [5], a multi-tier fog environment is introduced in which multiple implementations were integrated. The fog computing architecture in [4] is shown in Figure 1.1.

The fog computing architecture consists of three tiers:

1.1.1. Edge tier. This tier consists of service request generating user-devices such as mobile phones, wireless sensors and IoT devices. User-devices are characterized with limited computational capabilities. Moreover, user-devices are

mobile in nature as they can move from an area covered by a fog unit to another area that is covered by a different fog unit. Additionally, user-devices can be virtually clustered based on their geographical location.

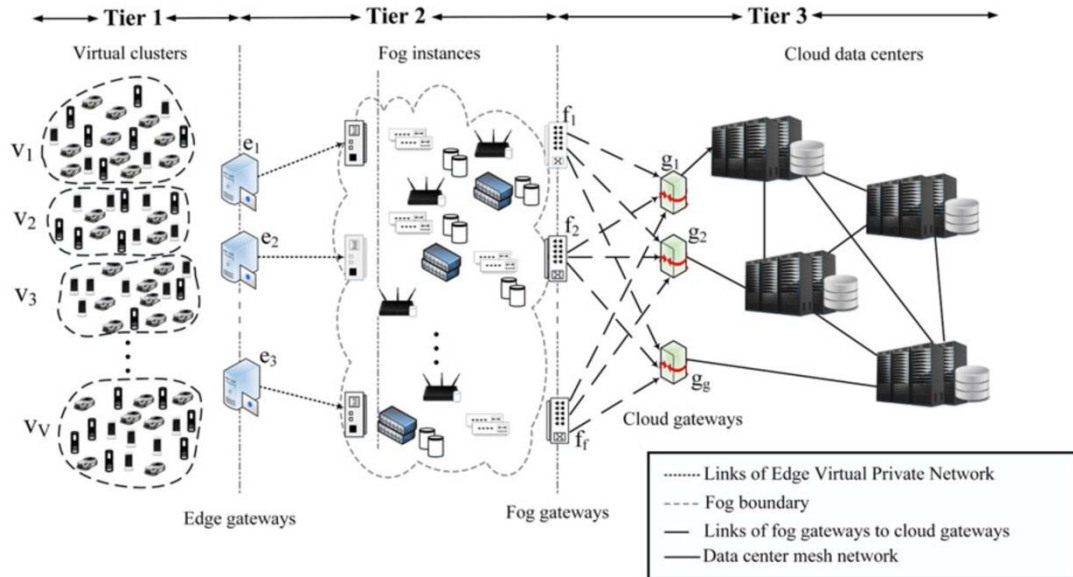


Figure 1.1: Fog computing architecture [4].

1.1.2. Fog tier. The network devices such as routers, switches, and access points are deployed as fog units within the network. These devices are virtually pooled to create self-governing devices, called fog instances, based on their geographical location. Therefore, fog instances are decentralized by nature. Moreover, the fog instances can be divided into two layers: abstraction and orchestration. Abstraction fog instances are responsible for resources management and virtualization, and for ensuring tenants' privacy. On the other hand, orchestration fog instances keep observing the state of the other fog instances and are responsible for providing fault tolerance techniques. In addition to the computational capability, orchestration fog instances contain storage devices that are temporarily needed while processing a service request.

1.1.3. Cloud tier. This tier comprises the geographically distributed cloud data centers that are responsible for providing the required services to user devices. The communication between cloud data centers is enabled through datacenter networks (DCNs) that are characterized with high bandwidth.

1.2. 5G Network

5G is the current generation of radio access networks (RAN). The transition to 5G is taken place by multiple internet service providers (ISP). For instance, Etisalat Telecom in the United Arab Emirates is ready to deploy the first 5G network in the Arab region [6]. Such a transition is motivated by two reasons: the exponential escalation in the number of users connected to the RAN and the capacity limitation of the current wireless spectrum [1]. The mm-wave technology is utilized in 5G networks by which the spectrum ranges from 3GHz to 300GHz. However, this technology is not enough to solve the aforementioned problems due to the characteristics of signal propagation and collision. In the literature, multiple solutions were introduced to allow maximum spectrum usability and avoid interference [7].

The architecture of a 5G network consists of two enabler technologies, the Cloud-RAN (C-RAN) and small cells. An environment that consists of both technologies is considered in [7]. The part of the RAN that connects its components, the C-RAN and small cells, is referred to as the fronthaul. On the other hand, the backhaul is the part of the network infrastructure that leads to the core network through which a connection to the internet and the cloud data center is enabled.

1.2.1. C-RAN. C-RAN is a centralized solution in which the base band units (BBU) and the remote radio heads (RRH) are separated [8]. BBUs are units that are responsible for signal processing while RRHs are the remote radio transceivers. All BBUs are centralized on-premise and virtualized. RRHs are distributed and are connected to the centralized pool of BBUs through a reliable wired medium, usually optical fibers [9].

1.2.2. Small cells. According to Cisco [10], small cells deployment is defined as a decentralized solution in which each small cell contains a BBU and an RRH that enable autonomous functionality. Small cells are deployed near the edge as part of the network, for example, as a local access point [11]. Small cells are different in their operational signal range and power consumption. Therefore, small cells are classified into three different categories. The categories from the most power-consuming and longest operational signal range to the least power-consuming and shortest operational signal range are: microcells, picocells, and femtocells [7].

1.3. Scheduling Problem

In General, the scheduling problem is concerned with situations where a set of events, jobs or tasks need to be allocated into a set of available shared resources [12]. In the literature, scheduling problems are commonly modeled as an optimization problem with various objectives and constraints. Objectives are involved with maximizing or minimizing specific criteria. Constraints are limitations that are captured from the environment in which the scheduling problem is given. Mainly, a scheduling problem consists of the following components [13]:

- *Requests*: a set of operations to be executed by resources. It can be logical or physical. Each operation has its own requirements.
- *Resources*: logical/physical elements provided with capabilities to execute requests according to their requirements.
- *Constraints*: rules and limitations that reflect the environment requirements and characteristics, i.e. precedence among requests and operations.
- *Objective*: it is the criterion for which the scheduling problem performance is evaluated.

1.4. Thesis Objectives

Although the adoption of 5G technology provides low-latency communication, relying on cloud datacenters to execute real-time service requests is inadequate due to the network delay in the backhaul connection. Fog computing offloads time-critical requests from the cloud servers, consequently avoiding sending requests through the backhaul connection. In 5G networks, small cells are good candidates for fog nodes deployment. In this work, the main objective is to minimize the latency of fog computing in 5G networks. This can be achieved by scheduling service requests, received by user-devices, across the small cells in the network such that the average makespan of all requests is minimized.

Furthermore, small cells are decentralized such that each small cell has its own knowledge and control, therefore capable of making scheduling decisions. The knowledge and control are distributed among private small cells that might belong to different groups of users that are rational. Hence, each small cell is considered an economically rational agent in which self-interest is the key individual behavior. Nevertheless, small cells need to share knowledge and compute capabilities to reach a

desired objective based on their interests. This goal-relevant interrelationship between small cells is referred to as interdependency [14]. The interdependency exists in the physical setup as small cells need to share their resources, and in the mental domain as small cells need to share their knowledge. Coordination between the small cells is required to enable their collaboration in order to achieve the ultimate goal of minimizing the latency. This work is concerned with developing a decentralized solution that provides a coordination framework to tackle the interdependency problem. Therefore, a collaboration between the small cells is enabled to reach the global objective of minimizing the latency of fog computing within 5G networks.

1.5. Research Contribution

The contributions of this work are as follows:

- A literature review is presented that covers the current challenges in fog computing and 5G networks, and the related solutions proposed by the research community.
- The scheduling problem is modeled as an optimization problem with the objective of minimizing the makespan considering the fog computing and 5G environment characteristics.
- The interdependency between the small cells is well defined and a novel decentralized framework is proposed to enable the coordination between the small cells to solve this interdependency problem.
- A game-theoretic approach is used to map the interdependency problem into a game which is modeled as a combinatorial auction problem using an economic approach.
- A novel decentralized heuristic solution is proposed to solve the decentralized scheduling problem in polynomial time.
- Model validations and system performance evaluation are presented to prove the adequacy of the proposed decentralized solution.
- Sensitivity analysis is presented in which the impact of each system parameter on the system behavior is investigated.

Chapter 2. Background and Literature Review

In this chapter, the fog computing challenges and suitability, and the 5G requirements and characteristics are reviewed. Moreover, the task scheduling problem is classified into two different types: centralized and decentralized scheduling problems. Furthermore, multiple criteria and performance metrics that are used in the literature to solve various scheduling problems are presented. In addition, game theory, and all components and classifications of games are discussed. Finally, previous works that provided a major contribution to solving problems related to fog computing and 5G challenges are investigated.

2.1. Fog Computing Challenges

Fog computing is a solution that extends the cloud capabilities nearer to the edge by which the communication delay is minimized. However, the deployment of such a solution brings multiple challenges. In this section, some of these challenges are presented.

2.1.1. Privacy and security. Fog computing can be deployed as opportunistic user-devices. This means that user-devices can share their resources to execute requests generated by other users in the network. In such an environment, incentive-based modeling is used in the literature to motivate network users to share resources and overcome privacy issues. However, the security of the opportunistic user-devices is still a concern [15].

2.1.2. Mobility. User-devices are mobile by nature. Therefore, moving from an area that is covered by a fog node to another area that is covered by a different fog node requires seamless handover techniques to ensure system continuity and reliability [16].

2.1.3. Interdependency. Fog nodes, that are considered autonomous agents, need to share knowledge and resources to achieve a global objective. This goal-relevant interrelationship between various agents is known as interdependency [14]. Thus, a coordination framework is needed to solve the interdependency problem by providing a way of collaboration between the autonomous entities.

2.1.4. Limited resources. Fog nodes are characterized by limited resources. Therefore, offloading is needed in which service requests are fairly distributed across all the fog nodes in the network. This is a scheduling problem where user fairness and network QoS is the main objective [17].

2.2. Suitability of Fog Computing

In addition to extending cloud capabilities nearer to the edge, fog computing is supposed to improve network latency, and reduce the overall usage cost and power consumption of the network. Considering these improvement aspects, the suitability and reliability of fog computing in satisfying the requirements of real-time requests were assessed by the work in [4]. The study considered a traditional network where fog nodes are implemented as network components, as shown in Figure 1.1. A simulation was conducted to compare a cloud-only environment with a fog/cloud environment considering multiple scenarios. The results showed that fog computing provides much lower latency considering low to normal user demands while the delay of fog computing and cloud-only environments were found to be the same considering high user demands. Furthermore, the study addressed the problem of CO₂ emission when relying completely on the cloud. Moreover, the experimentation outcomes confirmed that the fog/cloud environment results in less CO₂ emission by 59.26% and less mean power consumption by 42.2% compared to the cloud-only environment.

2.3. 5G Requirements and Characteristics

In this section, the requirements and characteristics of the 5G network are presented.

2.3.1. Enhanced mobile broadband. According to the 5G specifications, the data rate is defined to reach 10 Gbit/s in cellular networks. This is 10 times speed-wise enhancement compared to the 4G (LTE) network which is characterized with a data rate up to 1 Gbit/s [1].

2.3.2. Ultra-reliable and low latency communication. The fronthaul latency is expected to be in the range of sub-milliseconds to few milliseconds. For the backhaul connection, the goal is to keep the latency within the range of 1 to 10 milliseconds [9]. Furthermore, ultra-reliable connection of five nines, i.e. 99.999%, is required as the connectivity of 5G is anticipated to be always present [1].

2.3.3. Massive machine-type communication. One of the main 5G characteristics is to enable simultaneous communication of thousands of devices that are connected to the network. Such a requirement is essential since the number of user-devices connected to the network is exponentially increasing [1]. The Long Range Wide Area Network (LoRaWAN) is one of the currently available solutions that support mMTC [18]. The requirements and characteristics of 5G can be summarized in Figure 2.1.

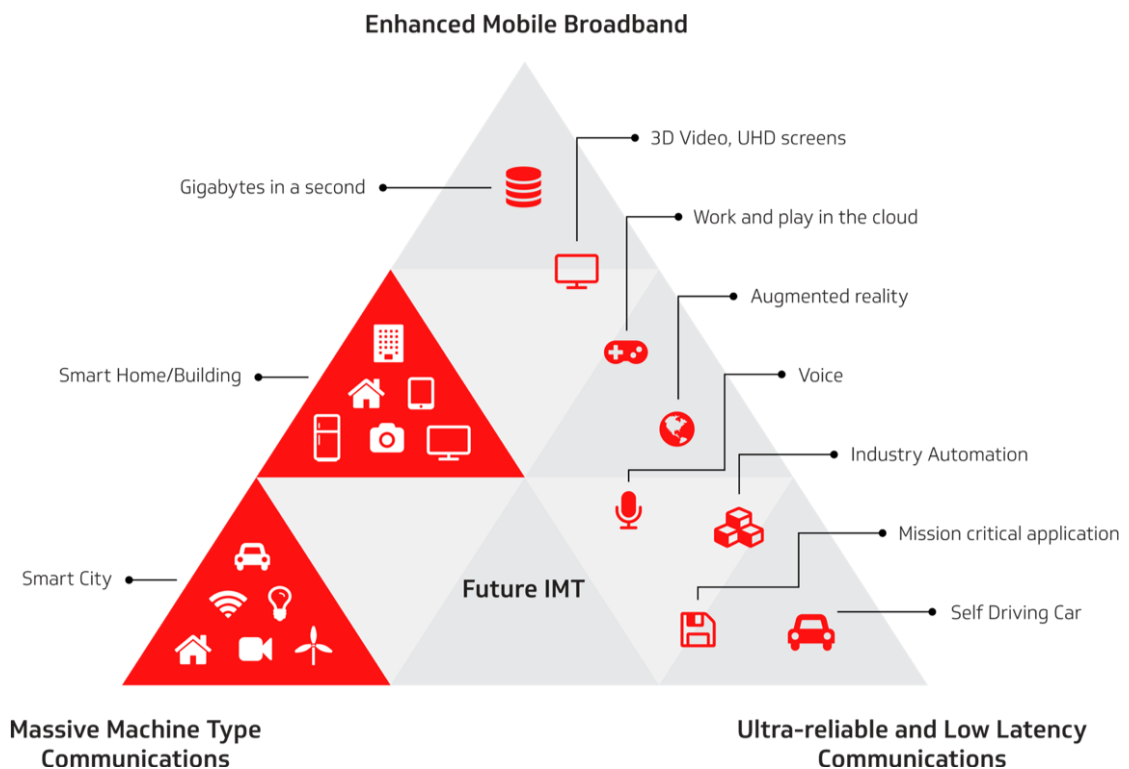


Figure 2.1: 5G requirements and characteristics [19].

2.4. Task Scheduling

Task scheduling problem is concerned with allocating shared resources to different tasks, i.e. service requests, such that a desired objective is achieved. As mentioned in Section 1.3, the scheduling problem is commonly modeled as an optimization problem. In addition, the nature of the environment influences the optimization model of the scheduling problem. Accordingly, an environment is categorized as a centralized or decentralized environment. Furthermore, the objective of the optimization model can be represented considering different performance

metrics. In this section, the scheduling problem considering a centralized or decentralized environment is described, and various performance metrics used in the literature are reviewed.

2.4.1. Centralized Scheduling. It is a scheduling problem in which a top-view scheduling process is executed by a central entity that has full knowledge of the environment settings and states. The central entity imposes decisions related to the control of the scheduling process. This setup has the advantage of achieving better results because of having complete knowledge regarding the involved entities and the schedule properties [20]. However, the centralized environment is associated with limitations related to scalability due to the high complexity of the scheduling problem. As a result, centralized solutions were found in the literature to be practical when considering small-scale scheduling problems [21].

2.4.2. Decentralized Scheduling. It is a scheduling problem in which the knowledge and the decision-making related to the control of the scheduling problem are dispersed across multiple autonomous entities. Each entity is considered a rational agent that is capable of taking its local decision and has its own knowledge of its local properties and states. Multi-agent systems are commonly used in the literature to model decentralized environments [22]. To solve the decentralized scheduling problem, a coordination mechanism is required to enable collaboration among agents in order to achieve a global objective. Although the theoretical results of solving scheduling problems considering a centralized environment are more promising than considering a decentralized environment, solutions for scheduling problems in decentralized environments are scalable as opposed to considering centralized environment settings [20]. However, there are multiple tradeoffs related to the decentralization. In decentralized scheduling problems, the knowledge is scattered. The solution accuracy and quality are degraded due to the lack of knowledge associated with each agent. Furthermore, the requirement of a coordination mechanism increases the complexity of the scheduling problem. Providing a coordination framework enables agents to collaboratively achieve a global objective. Nevertheless, such a collaboration increases communication overhead.

2.4.3. Complexity. The time complexity of solving a scheduling problem depends on its nature whether it was centralized or decentralized. In a centralized

scheduling problem, the complexity is related to the scheduling problem components, discussed in Section 1.3. This is referred to as the scheduling domain complexity. On the other hand, the complexity in a decentralized scheduling problem is associated with the distribution of knowledge and control across the multiple agents. Such complexity is referred to as knowledge distribution complexity and control distribution complexity. As agents need to share their knowledge, the knowledge distribution complexity leads to communication overhead due to message passing and data transmission between agents. Moreover, as mentioned in Section 2.4.2, a coordination mechanism is needed to solve the interdependency problem between agents. Agents' coordination consists of various components by which increases the complexity to the scheduling problem. One of the components is the interaction structure that defines the pattern of communication between agents. The design of the interaction structure impacts the problem complexity. For instance, minimizing the required amount of message passing between agents, reduces the communication complexity. Although the interaction structure can be imposed, each agent is an autonomous entity that takes its own decision related to the scheduling control, referred to as actions. In addition, each agent has its own preferences by which actions are chosen. In economics, agent preferences are quantified as utility values. As part of the coordination mechanism, each agent values its own preferences. The valuation process results in additional complexity. Further complexity results from choosing appropriate actions by agents based on utilities that reflect their preferences. Finally, the coordinated control related to the decision-making of the decentralized scheduling problem is referred to as the winner determination problem. Solving the winner determination problem produces more complexity to the scheduling problem.

2.4.4. Common Performance Metrics. A scheduling problem can be modeled as an optimization problem. The performance of a scheduling approach mainly depends on the objective function of the optimization model. The objective function can be formulated using different performance metrics. In this subsection, some of the most common scheduling performance criteria are presented.

2.4.4.1. Completion Time. This criterion is related to the time in which the execution of a request is completed. The corresponding objective function can be formulated as minimizing the sum or average of all request completion times. Assigning

weights to different requests are also possible. In the literature, minimizing the maximum completion time of all requests is also considered, which is known as minimizing the makespan [23].

2.4.4.2. Flow Time. This performance metric is related to the time during which the request is being executed in the system. Formally, the flow time of a request is the difference between the completion time and starting time of that request's execution. The corresponding objective function is formulated such that it minimizes the sum or average of request flow times. Weight assignments are used to prioritize different requests. Minimizing the maximum flow time is also considered in the literature [23].

2.4.4.3. Earliness. A request's earliness is the difference in time between the completion time and deadline of that request. If the request completion time passes the defined deadline, the request is referred to as late request and the corresponding objective value is represented as a negative value. However, a positive objective value indicates that the execution of a request is completed before the defined deadline. Maximizing the average or sum of earliness of all requests is a common objective function used in the literature [23].

2.5. Mechanism Design and Game Theory

In game theory, mechanism design is the creation and crafting of game rules that impact player interactions so that a desired optimal or equilibrium outcome is achieved [24]. In mechanism design, instead of modeling an environment using a game-theoretic approach, game rules are designed according to a desired outcome. Therefore, mechanism design is known as "reverse game theory". In the literature, game-theoretic approaches were used to model scheduling problems given decentralized environment settings. Furthermore, game components and classifications are discussed in this section.

Game Theory is the study of modeling problems that involve interactive choices between autonomous entities with global objectives. It describes mathematically the behavior of each entity and the interaction among them [25]. The components of any game can be summarized as follows:

- *Players*: a set of self-governing entities that are involved in the game.
- *Strategies*: a set of possible actions that describe the players' decision-making.

- *Payoff*: a set of outcomes where each outcome corresponds to a combination of strategies.
- *Equilibrium*: the state in which all players have no desire of changing their strategies, considering other players' decisions.

Moreover, games are classified based on the nature of the problem environment. In addition to the aforementioned components, game classifications reflect problem characteristics that impact the modeling process. Some of these classifications are discussed in this subsection.

2.5.1. Non-cooperative vs. Cooperative Games. Game theory is mainly classified into non-cooperative and cooperative game theories. The non-cooperative game theory studies situations that involve rational selfish players, economic agents, in which maximizing self-payoff is the focal goal. It analyzes the possible strategies at an individual level such that it describes and covers the finest game details. In this category, the global objective is to reach a state in which any individualistic strategy deviation taken by any player has no payoff gain. This state of a game is known as Nash Equilibrium. On the other hand, the cooperative game theory focuses on the analysis of group formations, known as coalitions. The main objective in a cooperative game is to build a game structure by which players' cooperation is preferred over individuality. The coalition that contains all players is called the grand coalition. In a cooperative game, reaching the state in which the grand coalition is the most preferred choice for all the players is equivalent to achieving the equilibrium in a non-cooperative game. Cooperative games describe payoffs and strategies as coalitions and groups. Hence, cooperative game theory describes games at an abstract level as opposed to the non-cooperative game theory. Therefore, cooperative games can be converted into and be expressed as non-cooperative games, however, the opposite does not hold [26].

2.5.2. Sequential vs. Simultaneous Games. In game theory, player behaviors are modeled as sets of potential actions that define player strategies. In sequential games, player choices are dynamically affected by other players' previous actions. By contrast, player strategies in simultaneous games are completely independent of other players' past decisions. In other words, sequential games, as opposed to simultaneous games, effectively involve the time factor. The term "effectively" is used as the time axis existence alone with no impact on player behaviors classifies the game to be

simultaneous. For example, if players are to play in turn sequentially but none of them knows the actions taken by the other players, then the time factor has no strategic effect. Consequently, the game is considered to be an effectively simultaneous game. Games that are repeated in multiple stages, known as repeated games, are considered to be sequential if the players in a given stage exploit information regarding actions taken by other players in a previous stage. On the other hand, a repeated game with completely independent stages is an example of effectively simultaneous games [27].

2.5.3. Information in Games. In game theory, games are classified based on the knowledge shared among all players into complete or incomplete information games, and perfect or imperfect information games. In a complete information game, all players share the data of each other's payoffs, utility functions, and strategies. On the other hand, players in perfect information games have full knowledge of other players' current and previous states in the game, in the case of sequential games, as well as the initial state of all players. Games which are complete information are not necessarily perfect information, and the opposite is also true. Simultaneous games are usually considered as imperfect information games because each player's next choice is secretive from the others while such games are considered complete information games as strategies and outcomes are known to all players, for example, rock-paper-scissors [25].

2.5.4. Combinatorial Games. In computational complexity theory, combinatorial problems are optimization problems in which the decision-making process involves searching for an optimal solution in a considerably large search space of feasible solutions such that it is untraceable. In game theory, combinatorial games involve players which have an enormous number of possible actions such that finding an optimal strategy is not computable in polynomial time. Games with imperfect information are normally considered as combinatorial games since the lack of information extends the possibilities of all player actions. Solving games of this category is related to solving non-deterministic polynomial (NP) optimization problems [28].

2.6. Related Work

Many challenges related to the fog computing paradigm and the 5G technology are studied in the literature. Fog computing elevated problems of interdependency,

mobility, load balancing, and privacy and security. In 5G networks, challenges such as interference and power offloading are considered. In this section, state-of-the-art solutions proposed by the research community to address some of the challenges related to fog computing and 5G.

A solution for the interference problem in 5G networks was proposed in [7], considering a dense deployment of small cells in 5G RAN. The network model used is orthogonal frequency-division multiple access (OFDMA) multi-tier dense heterogeneous networks where the spectrum is divided into multiple carriers for multiplexing. Furthermore, subcarriers are assigned to users for multi-user access. Accordingly, the smallest resource unit that can be allocated to one user is called Physical Resource Block (PRB). A PRB is characterized by a predetermined fixed band of specific time and frequency. For example, a PRB can have a fixed band of 1 ms and 200 kHz. Small cells are divided into three types according to their area of coverage: microcells, picocells, femtocells from largest to smallest coverage, respectively. Small cells are used for base stations' job offloading. Moreover, the authors assumed a C-RAN as a central entity that gathers all information about the network and makes decisions accordingly. Interference was modeled as an optimization problem with the objectives of maximizing fairness and spectrum reuse, and minimizing interference. QoS is represented as assigned weights corresponds to each allocation. The proposed NP-hard optimization model is mapped to Linked Conflict Graph (LCG) that consists of vertices and edges where vertices represent the connection between user-small cell pairs and edges represent conflict, potential interference, between vertices. Subsequently, a state-of-the-art centralized algorithm was proposed to solve the LCG problem. Due to the high complexity of the exhaustive search involved in the centralized solution, a heuristic approach is proposed for faster algorithm execution that is relying on assigning PRBs to the nodes probabilistically. Results showed that the proposed algorithm achieved 26% and 16% improvement compared to other well-known algorithms in the literature.

The work in [17] considered the problem of load balancing in fog computing that involves interoperating small cells that collaborate to satisfy user requests. An edge cloud computing environment is considered which consists of small cells of type femtocells that have a transmission range of a department and usually owned by a

specific network user. All femtocells are provided with computational and storage resources. The work proposes a centralized solution of a Small Cells Manager (SCM) that gathers information about the network as required for later decision-making, consequentially enabling network orchestration. The air interface connects all femtocells via point-to-point links with the OFDMA setup in which the spectrum is divided into multiple carriers for multiplexing. The author assumed multi-user multi-cell environment settings such that multiple users are served by multiple small cells. The main objective of this work is to distribute the workload among interconnected small cells. The proposed solution includes creating small cell clusters by an SCM such that all resources in a cluster are pooled to serve requests generated by user-devices connected to any of the composing femtocells. Clusters are formed and changed dynamically based on resource availabilities. An SCM is responsible for allocating the required resources for each service request according to a user-defined objective and performance metric. Accordingly, the authors proposed a customizable generic algorithm that can be changed according to a desired objective. Three variations of the proposed generic algorithm are considered. First, the Earliest Deadline First – Power Consumption (EDF-PC) that aims to minimize the power consumption while using EDF as the scheduling rule. Second, the Earliest Deadline First – Latency (EDF-LAT) in which the main goal is to minimize the cluster overall latency using the EDF scheduling rule. Finally, the third proposed algorithm variation is the Computational Size – Latency (CS-LAT) where minimizing the latency is the objective and the scheduling rule depends on the computational size of a request. The work compared the three approaches considering three aspects: user satisfaction ratio, average latency-gain per service request and average power consumption per service request. The first observation is that by increasing the number of users per small cell, the user satisfaction ratio of no cluster option drops dramatically. The static cluster has overall low-performance results that confirm the need for dynamic cluster formation to enable system adaptivity and orchestration. Considering up to 4 simultaneous users, EDF-LAT and EDF-PC algorithms maintained 95% of user satisfaction while CS-LAT only achieved 90%. However, CS-LAT algorithm accomplished the highest latency gain because all local available computational resources are allocated to satisfy the request locally and latency is considered while forming small cell clusters. On the other hand, EDF-PC utilizes the allowed time by allocating minimal compute resources to each

request, consequentially achieving the balance between the users' satisfaction and power efficiency. The results showed that CS-LAT consumes less transmission power than EDF-LAT as the former executes more requests locally, therefore forwarding fewer requests to other small cells in the cluster.

The work in [5] proposed a multitier fog computing framework to involve large-scale big data analytics within the fog computing paradigm. Most of the existing fog computing architectures are based on radio access networks where enough computational power is provided in fixed network devices, i.e. small cells or routers. However, such a setup offers no flexibility when considering on-demand deployments. In [5] a fog-based cloud IoT environment was considered that consists of a layer of ad-hoc fogs, combinations of opportunistic user-devices. Opportunistic devices are user-devices such as smartphones, laptops, and vehicles that can be used as fog nodes whenever possible. Such devices are characterized by minor computational capability. Ad-hoc fogs are self-governing autonomous devices which makes this environment to be decentralized. Furthermore, the proposed environment consists of another tier of dedicated fogs, network fixed pre-deployed small cells or base stations. A cloud computing end is assumed in this work where datacenters execute requests with more complex processes if the two fog layers, dedicated and ad-hoc fogs, cannot satisfy the request requirements. Fog nodes are divided into ad-hoc and dedicated fogs according to their physical model. In addition, authors divided fog nodes according to their functional model into two types. The first type is master fogs of which the objectives are job scheduling, service management, and ad-hoc fog cluster creation. The second type is worker fogs that can accept invitations from master fog devices to be part of the ad-hoc cluster and share resources. Worker fogs are responsible for their self-management and consumption reporting to their master fogs. The work in [5] represented the functions of fog devices as modules. First, the service module provides data analytics services using distributed computing engines. Second, the job management and scheduling module which enables the communication between master fog devices to distribute the job among workers in order to get the best service response time. Third, the fog management and resource allocation module in which master fog devices create fog clusters by inviting worker fogs and allocate their available resources to jobs according to the best completion time. Additionally, the life cycle and security of worker fogs are handled by master fogs in this module. Fourth, the network and

virtualization module where master fog devices are adaptive to network changes to provide the best QoS. Virtualization is optional but preferable so that the privacy and security of worker fogs are enhanced. Fifth, the resource module is related to the data-generating devices, and the available computing and networking resources. The focus of this work is to improve the QoS in large-scale big data analytics across fog unites in the proposed decentralized environment considering real-time requests. The authors developed a QoS aware job and resource management schemes. In addition, a benchmark is proposed for evaluation purposes. Furthermore, a workload model is created from the benchmark by using sample jobs. This model is initially created before using the system. Afterward, the workload model is updated while using the system by getting feedback generated online after every job is executed successfully. Simulation for evaluating the proposed multitier fog framework with QoS aware service and job management is conducted. By comparing environments of ad-hoc fog only, ad-hoc fog with dedicated fog, cloud-only and all combined, the author emphasized that cloud-only approach is not enough to satisfy the requirements of real-time requests. Moreover, ad-hoc fogs cannot handle large-scale data analytics without the aid of dedicated fogs. Furthermore, the experiments showed that combining ad-hoc fogs, dedicated fogs as well as the cloud led to promising results. Finally, the monetary revenue of using the proposed multitier fog framework is found to be double the revenue of the cloud-only based environment.

Power consumption is a concern in 5G networks as IoT devices usually rely on batteries as the main power source. Accordingly, developing an energy-efficient communication framework seems to be necessary. In [29], the power consumption problem is mapped to a resource allocation problem. The work proposes a solution for the resource allocation problem with the objective of minimizing power consumption in a 5G network considering full-duplex communication between a central controller, and multiple sensors and actuators. The environment considered consist of a physical layer of sensors and actuator, a communication network layer of which the spectrum is divided into equal bandwidth channels, and a central controller in which the analysis of the collected data and the consequent decision-making are performed. Communication can operate in a full-duplex or half-duplex mode on each channel. The spectral resource is divided into channels with equal bandwidth. Digital and analog cancellation technologies are assumed to be used to avoid self-interference in full-duplex

communications. Each channel can be allocated to one sensor and one actuator at any given time and each device can communicate with the central controller through one channel only. First, the problem is formulated as an optimization problem with the objectives of minimizing power consumption and maximizing the transmission rate. The author assumed a one-to-one connection between channels and sensors, and between channels and actuators. The concept of virtual devices, i.e. devices that have zero gain and no efficiency, was introduced by the author for generalization purposes. The optimization problem is solved using mixed-integer nonlinear programming as it has continuous and binary variables. Furthermore, the problem of resource allocation was divided into two sub-problems: the power allocation and channel allocation problems. The power allocation problem is related to optimizing channel transmission power. The channel allocation problem is concerned with optimizing the allocation of channels across devices, sensors and actuators. In [29], three cases were considered to achieve the best power utilization which are: the case of real sensor and virtual actuator, the case of virtual sensor and real actuator, and the case of real sensor and actuator. For the first two cases, the optimization problem is a rational-nonlinear function, therefore, it is converted into a convex optimization problem. Accordingly, Dinkelbach's algorithm was used to iteratively varying transmission power starting from the max power until it converges. For the third case, game theory was used such that sensors and actuators are considered as the player of a non-cooperative game. Players are selfish economic agents that aim to maximize their payoffs, efficiencies, and have strategy sets that depend on the max transmission power. The behavior of each device, sensor or actuator, is modeled as an optimization problem in which the transmission power of a device depends on the transmission power of the other device. By using Dinkelbach's algorithm, the optimization problem of each device is solved separately and iteratively using the last calculated transmission power of the other device until the Nash equilibrium is reached. Moreover, the channel allocation problem was modeled as a 3-D matching problem. According to the proposed solution, the 3-D matching problem is reduced into 2-D matching problem by matching two of the three sets arbitrarily. Subsequently, the Hungarian method was used to solve 2-D matching problems iteratively such that the matched pairs are changed in every iteration. For system evaluation, comparisons were held among different algorithms. First, the proposed Iterative Hungarian Method with Virtual Devices (IHM-VD). Second, the greedy

algorithm where each channel selects the most efficient sensor-actuator pair. Third, an exhaustive search that achieves the optimal solution but with high computational complexity. Forth, the Two-Sided Hungarian algorithm (TSH) that converts the 3-D matching problem into two 2-D problems such that each 2-D problem deals obtain the best allocation for channel-sensors and channel-actuators separately. Fifth, the Iterative Hungarian Method without virtual devices (IHM). Sixth, the Half-Duplex Resource Allocation (HDRA). Results showed that both proposed algorithms and HDRA performs the same. Moreover, results demonstrated that the efficiency eventually saturates for IHM and greedy algorithms while it keeps increasing for the two proposed algorithms and TSH. The performance of IHM-VD is much higher than TSH because TSH does not consider the efficiency between the sensor-actuator pairs. Furthermore, the results proved that IHM-VD always converges to a solution. Although the exhaustive search algorithm outperforms IHM-VD, the proposed algorithm is much simpler considering the complexity of the problem.

To conclude, multiple challenges related to fog computing and 5G are studied in the literature. Moreover, many research works contributed to solving task scheduling problems that involve allocating decentralized resources [22]. In addition, the scheduling problem of allocating service requests across multiple fog nodes to minimize service latency is solved by assuming a centralized entity that is capable of enforcing control decisions related to the scheduling process [30]. The same scheduling problem is studied considering the 5G environment in [31]. However, a centralized manager, called master F-RAN, is assumed to be able to control the resource allocation of all the fog nodes in the 5G network. To the best of our knowledge, the proposed solutions in the literature did not address the interdependency problem between the decentralized fog nodes within the 5G network. Hence, this work focuses on the problem concerned with scheduling service requests across decentralized fog nodes within the 5G network with the objective of minimizing the overall latency. Furthermore, the interdependency between the fog nodes is considered in this work and a coordination framework is proposed to handle the interdependency problem.

Chapter 3. Scheduling Problem Modeling

As mentioned in Section 1.4, this research work is concerned with the scheduling problem of allocating fog computing resources to the requests generated by user-devices in 5G networks with the objective of minimizing the overall latency. In this chapter, the fog computing environment in a 5G network and its characteristics are presented. Furthermore, the system and all its components are modeled. In doing so, several latency factors are considered. The scheduling problem is modeled as an optimization problem with the objective of minimizing the makespan. Eventually, the proposed system model is validated using hand calculation in conjunction with the use of an optimization engine.

3.1. Environment Overview

In 5G networks, combining C-RAN and small cells allows the implementation of collaborative technologies by which the distributed small cells jointly satisfy user requests. Accordingly, low-latency communication among network components is provided to help achieve the objective of minimizing the overall latency. The idea of multi-tier fog computing architecture is supported by many researchers in the literature [5], [30]. In such an environment, the C-RAN is connected to multiple small cells. Each small cell is connected to several user devices. The C-RAN has computational power to serve as a manager that can execute collaborative algorithms. Moreover, the C-RAN contains a pool of BBUs that are virtually provisioned to small cells on demand. BBU functions are only provided by the C-RAN. Therefore, each small cell will serve as a fog node that contains RRH and computing resources. A similar network architecture is deployed in a solution available in the market, known as OneCell [32]. The architecture of fog computing in a 5G network is presented in Figure 3.1.

3.2. Modeling Resources

In each radio access network, a C-RAN is connected to a set of I small cells, fog nodes. Each small cell is denoted as i where $1 \leq i \leq I$. Moreover, each small cell i has a computational power capacity P_i^{max} that is divided into P_i virtualized compute blocks with compute capability of ΔP instructions/sec. Compute blocks of a small cell are a logical representation of the small cell's physical compute capability. A compute block is denoted as p where $1 \leq p \leq P_i$.

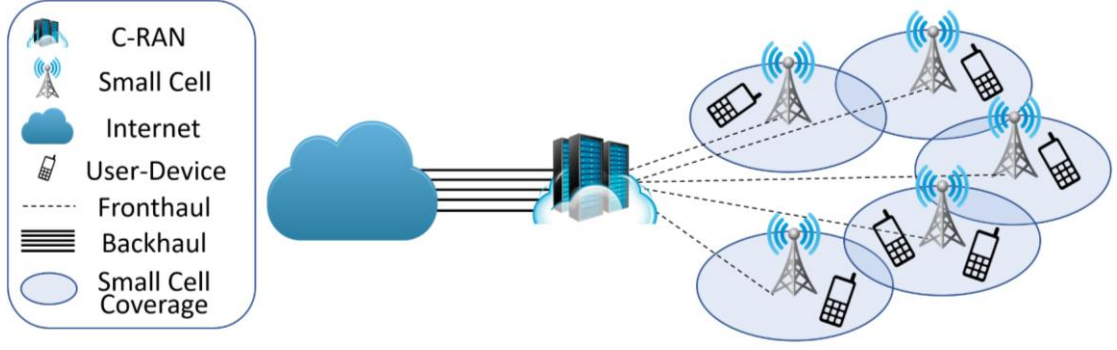


Figure 3.1: Fog computing/5G architecture

3.3. Modeling Time Dimension

The allocation of requests shall be determined for a specified time window T . Time is discretized into time slots t where $0 \leq t \leq T$ and $T = t^{max}$ is the time slot in which the deadline of the last executed request is defined. The duration of a time slot t is noted as t_d which depends on the number of instructions being executed by a compute block per time slot. All requests are assumed to be ready at time $t = 0$.

3.4. Modeling Service Request

Small cells are connected to several user-devices that generate a set of service requests. Each request is denoted by j where $1 \leq j \leq J$ and J is the number of service requests generated by all user-devices connected to the RAN. Each request j is characterized by a size N_j in bits and a deadline τ_j in seconds. Additionally, a service request consists of multiple operations to be executed, i.e. $j = \{O_{j1}, O_{j2}, \dots, O_{jK_j}\}$ where K_j is the number of operations that need to be executed to fulfill request j and $1 \leq k \leq K_j$. Each operation is described by instruction count P_{jk} , a binary variable $E_{jkt} = 1$ if operation O_{jk} is executed at time t and $E_{jkt} = 0$ otherwise, and a starting time slot t_{jk}^s . For a request to be served, all its operations must be completed. Furthermore, there are no dependencies among different requests. However, operations might depend on the outcomes of other operations of the same request. This can be represented as follows:

$$\gamma_{jkh} = \begin{cases} 1, & \text{if } O_{jk} \text{ depends on } O_{jh} \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

$$0 \leq \gamma_{jkh} + \gamma_{jhk} \leq 1, \quad \forall j \text{ and } 1 \leq k, h \leq K \quad (2)$$

γ_{jkh} is a binary variable that indicates if there is a dependency between operations O_{jk} and O_{jh} . Equation (2) prevents two-way dependency between any two operations to avoid deadlock. For further clarification, Figure 3.2 represents the workflow of an example request.

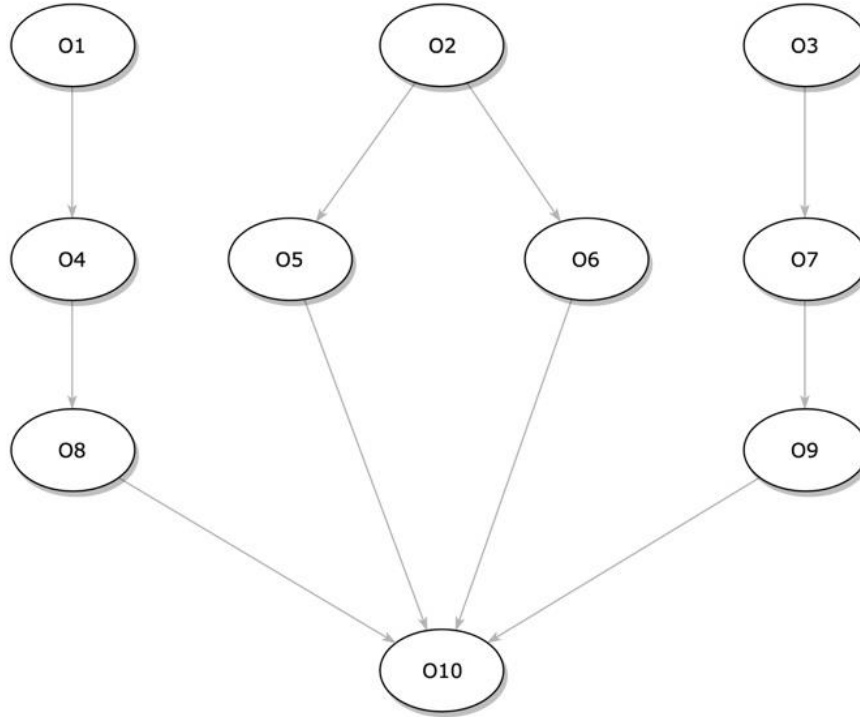


Figure 3.2: The workflow of an example request

In Figure 3.2, the completion of operation O_2 is required to start the execution of operations O_6 and O_5 . Parallel operations such as operations O_1 , O_2 , and O_3 can be executed concurrently. In this example, operation O_{10} can only be executed after performing all other operations. This can be an aggregation of outcomes of all other operations. Moreover, a path can be defined as a series of dependent operations starting from the first independent operation to the last dependent operation. In Figure 3.2, there are 4 paths: $\{O_1 \rightarrow O_4 \rightarrow O_8 \rightarrow O_{10}\}$, $\{O_2 \rightarrow O_5 \rightarrow O_{10}\}$, $\{O_2 \rightarrow O_6 \rightarrow O_{10}\}$ and $\{O_3 \rightarrow O_7 \rightarrow O_9 \rightarrow O_{10}\}$. In addition, a dependency level can be defined as a level in which all operations can be executed concurrently. In this example, 4 dependency levels can be recognized: $\{O_1, O_2, O_3\}$, $\{O_4, O_5, O_6, O_7\}$, $\{O_8, O_9\}$ and $\{O_{10}\}$.

3.5. Modeling Dependency

Let operation k depend on operation h of a request j , i.e. $\gamma_{jkh} = 1$. To start the execution of operation k , operation h should be completed first, i.e. $E_{jht} = 1$. On the other hand, if both operations are independent, then each operation can be executed regardless of the value of E_{jkt} or E_{jht} . Thus, such a relationship can be described in the Boolean expression presented in (3).

$$\neg(\gamma_{jkh} \wedge \neg E_{jht}) \quad (3)$$

The Boolean expression in (3) describes the feasibility of executing operation O_{jk} according to the dependency constraint. This relation can be proved by a truth table represented in Table 3.1:

Table 3.1: Truth table shows dependency relationship

γ_{jkh} (O_{jk} depends on O_{jh} ?)	E_{jht} (is O_{jh} executed ?)	$\neg(\gamma_{jkh} \wedge \neg E_{jht})$ (can O_{jk} be allocated?)
0	0	1
0	1	1
1	0	0
1	1	1

Table 3.1 shows that any operation O_{jk} cannot start being executed in a time slot t only if it depends on another operation O_{jh} that is not completed before the time slot t , such that $\gamma_{jkh} = 1$ and $E_{jht} = 0$. This conforms with the definition of operations dependency. The boolean relationship in (3) is expressed mathematically in (4).

$$|(\gamma_{jkh} * |E_{jht} - 1|) - 1| \quad (4)$$

In (4), the logical And (\wedge) is a simple mathematical multiplication while the unary operator Not (\neg) can be substituted with the following mathematical expression $\neg a = |a - 1|$, where a is a binary variable. To allocate a compute block to an operation, the dependency constraint must be satisfied. Hence, $x_{jk ipt}$ can only be 1 if and only if the dependency relationship in (4) equals to 1, otherwise $x_{jk ipt} = 0$.

3.6. Modeling Latency in 5G Environment

Latency is the delay in time between requesting a service and the consequent response. When a service request is initiated by a user, it is sent to the available service provider, a cloud datacenter or a fog node, for execution. Afterward, the corresponding response is sent back to the user. This delay in time is influenced by multiple delay factors.

3.5.1. Propagation delay. It is the delay that depends on the distance between the sender and the receiver, and the propagation velocity of the transmitting medium. The propagation delay is calculated as follows:

$$D_p = \frac{d}{V_p} \quad (5)$$

where d is the distance in (m) between the sender and the receiver and V_p is the propagation velocity in (m/s). The propagation velocity depends on the physical infrastructure. In the fronthaul of 5G networks, optical fibers are used as the transmitting medium. Accordingly, the propagation velocity, considering optical waves, is defined as:

$$V_p = \frac{1}{n} c \quad (6)$$

In equation (6), the variable n is the refractive index of the fiber core and c is the speed of light in vacuum. Optical fibers used in communication nets typically have a core refractive index of approximately 1.5 [33]. Accordingly, the propagation velocity of optical fibers is calculated as follows:

$$V_p = \frac{1}{n} c = \frac{1}{1.5} * (3 * 10^8) \approx 2 * 10^5 \text{ km/s}$$

Furthermore, the distance between a small cell and the C-RAN is up to 20 km [34]. Therefore, the maximum propagation delay is calculated as follows:

$$D_p = \frac{20 \text{ km}}{2 * 10^5 \text{ km/s}} = 100 \mu\text{s}$$

This is the maximum propagation delay caused in the fronthaul. However, the actual delay value depends on the distance between the C-RAN and the small cells.

3.5.2. Transmission delay. It is the delay in time during which the data is transferred from the sender to the receiver. The transmission delay depends on the size of the data being transferred and the transmission bit rate of the communication link. The transmission delay in (s) is calculated as presented in (7).

$$D_T(j) = \frac{N_j}{R} \quad (7)$$

N_j is the data in (bits) of request j and R is the transmission rate in (bits/s). The transmission rate depends on the implementation of the C-RAN. The Common Public Radio Interface (CPRI) protocol is commonly used to handle the communication between the small cells and the C-RAN [35]. According to CPRI specifications, the protocol has a fixed transmission bit rate. For flexibility purposes, there are 10 options available, each with a different bit rate. Option 1 offers 614.4 Mbit/s while option 10 offers bit rate up to 24.33 Gbit/s [36].

3.5.3. Compute delay. It is the time needed to fully execute a service request. It depends on the instruction count of the request to be executed and the computational power of the processing unit. The compute delay of a request can be defined as the makespan of that request. The makespan of a request is the execution time of the request's last operation in addition to the starting time of that operation. In other words, the makespan of a request is a measure of the time from which the request is ready until all its operations are completed. Therefore, the makespan can be calculated using the equation presented in (8).

$$C_j = \max_k \left(t_{jk}^s * t_d + \frac{P_{jk}}{\sum_i \sum_p^{P_i} x_{jkip, t_{jk}^s} * \Delta P} \right), \quad \forall j \quad (8)$$

C_j is the maximum completion time of all operations of request j , i.e. the makespan of request j . t_{jk}^s is the starting time slot of operation O_{jk} and t_d is the time duration of each time slot in (s). P_{jk} is the instruction count of operation O_{jk} . $x_{jkip, t}$ is a binary variable where $x_{jkip, t} = 1$ if compute block p in small cell i is allocated to execute operation k of request j at time t , and $x_{jkip, t} = 0$ otherwise. Notations for the aforementioned equations are summarized in Table 3.2.

Table 3.2: Summary of notations

Notation	Explanation
i	Small cell/ fog node, $1 \leq i \leq I$
j	Service request generated by a user, $1 \leq j \leq J$
N_j	Size of request j in bits
τ_j	Deadline of request j
t	Time slot, $0 \leq t \leq T$, and $T = t^{max}$
t^{max}	Time slot in which the last request deadline is defined
t_d	Duration of each time slot
k	An operation, $1 \leq k \leq K_j$, where K_j is the number of operations in request j
O_{jk}	Operation k of the request j
t_{jk}^s	Starting time slot of O_{jk}
E_{jkt}	Equals 1 if O_{jk} is executed before time t , 0 otherwise
γ_{jkh}	O_{jk} depends on O_{jh} , $1 \leq k, h \leq K_j$
D_p	The propagation delay of j
$D_T(j)$	Transmission delay of j
C_j	Makespan of a request j
p_i^{max}	Compute capability of i
p	Compute block p in small cell i
ΔP	instruction/sec of each CB
P_{jk}	Instruction needed to complete an operation k of a request j
$x_{jk ipt}$	Allocation of a CB p in a fog i to an operation k of a request j at a time t

3.7. Problem Formulation

For the considered system architecture, multiple assumptions are presumed to emphasize the environment characteristics. The assumptions are listed below:

- Small cells execute critical service requests only. Accordingly, all non-critical requests are forwarded to the cloud.
- Any user device can be connected to only one small cell at a time.
- No dependencies among requests. However, each request consists of multiple dependent or independent operations.
- All operations are non-preemptable.

In 5G networks, small cells are usually provided by limited computational power. In addition, the generated service requests are nonuniformly distributed across the small cells in the network. Therefore, all requests need to be scheduled for execution by considering all small cell resources such that the overall latency is minimized. The total latency is influenced by multiple delay factors, discussed in Section 3.6. The propagation and transmission delays are fixed factors related to the underlying network infrastructure. The compute delay, on the other hand, depends on the system decision-making to allocate small cell resources to requests' operations. This is a scheduling problem that aims to minimize the computational delay. The scheduling problem objective function can be formulated by considering different criteria and performance measures [23]. In the literature, the makespan is the most common criterion considered when minimizing the latency is the main objective [37]. Thus, the scheduling problem is modeled as an optimization problem, presented in equations (9-15):

$$\min_{x_{jk ipt}} \sum_j^J C_j \quad (9)$$

s.t.

$$\sum_j^J \sum_k^{K_j} x_{jk ipt} \leq 1, \quad \forall i, \forall p, \forall t \quad (10)$$

$$\sum_i^I \mathfrak{S}_i(O_{jk}) \leq 1, \quad \forall j, \forall k \quad (11)$$

$$x_{jk ipt} \leq |(\gamma_{jkh} * |E_{jht} - 1|) - 1|, \quad \forall i, \forall j, \forall k, \forall h, \forall t \quad (12)$$

$$C_j + D_T(j) + D_p \leq \tau_j, \quad \forall j \quad (13)$$

$$|x_{jk ip, t} - x_{jk ip, t+1}| = |E_{jk, t} - E_{jk, t+1}|, \quad t_{jk}^{start} \leq t < T, \forall j, \forall k, \forall i, \forall p \quad (14)$$

$$\sum_i^I \sum_p^P \sum_t^T x_{jk ipt} \geq \frac{P_{jk}}{\Delta P}, \quad \forall j, \forall k \quad (15)$$

The optimization problem, given in equations (9-15), is a Mixed-Integer Quadratically-Constrained Programming (MIQCP) problem. The objective function presented in (9) is subject to constraints presented in equations (10-15). Constraint (10) indicates that a compute block can be allocated only to one operation at a time. Constraint (11) ensures that any operation can be executed by only one small cell at a time. The operations of a request can be distributed for execution by multiple small cells and each operation can be allocated to many compute blocks. However, considering the geographical distribution of small cells, all compute blocks that are allocated to an operation must be located in the same small cell. This constraint is to prevent operation communication overhead. Such a constraint is modeled as follows:

$$\mathfrak{S}_i(O_{jk}) = \begin{cases} 1, & \sum_p^P \sum_t^T x_{jk ipt} > 0, \\ 0, & \text{otherwise} \end{cases} \quad \forall i, \forall j, \forall k \quad (16)$$

In equation (16), $\mathfrak{S}_i(O_{jk}) = 1$ if small cell i is processing operation O_{jk} , and $\mathfrak{S}_i(O_{jk}) = 0$ otherwise. Equation (12) represents the request's dependency constraint. Constraint (13) shows that the total delay of any request must be less than the deadline requirement of that request. Constraint (14) ensures that any compute block allocated for any operation shall not be freed until all the corresponding instructions are executed, which is required for the non-preemption assumption of operations. The last constraint, presented in equation (15), guarantees the allocation of the minimum number of resources required by each operation.

3.8. Model Validation

Multiple parameters need to be set to validate the proposed model. Some of the parameters reflect the environment such as the number of small cells, and the number of requests and their operations. Other parameters reflect the settings that can be modified by the system administrator, i.e. ISP, such as the number of compute blocks and their capabilities which also affect the duration of each time slot. Table 3.3 summarizes all system parameters considered in the validation process.

For validation purposes, the following example is considered that consists of three requests where each consists of three operations. Requests have hard deadlines to

be met and operations are characterized by numbers of instructions that must be fulfilled. Table 3.4 provides information regarding the requests and their operations of the considered example.

Table 3.3: Summary of system parameter values

Parameter	Value
# of fog elements	2 Fogs
# of CBs per fog	5 Compute blocks
Fog CPU speed	1 GHz
Fog Performance	1.3 Cycle Per Instruction
CB CPU speed (Cycles)	200 MHz
CB CPU speed (ins/sec)	Approx. 150 MIPS
Duration of each time slot	666.7 ms
CB CPU speed (Instruction/time slot)	0.1 MIPS

Table 3.4: Requests information

Request j	Instruction count ($\times 10^5$) IPS For operation k			Deadline (time slot)
	$k = 1$	$k = 2$	$k = 3$	
$j = 1$	2	4	5	$t = 3$
$j = 2$	3	2	4	$t = 3$
$j = 3$	4	4	3	$t = 3$

Some dependencies exist among operations of some requests. Dependencies can be represented in a matrix form as shown in Table 3.5, where each sub-table reflects the dependencies among the operations of a request.

Table 3.5 consists of three sub-tables. Each sub-table provides information regarding dependencies among operations of a request. For example, the first sub-table shows the dependency information among the operations of the request $j = 1$. Each sub-table can be interpreted as follows: operation k depends on operation h only if the value of the intersectional cell is “1”. The value “0” indicates that operation k does not

depend on operation h . For example, in Table 3.5, operation 2 depends on operation 1 and operation 3 depends on operation 2 for all requests.

Table 3.5: Dependency information 1

Agent i	Dependency of O_{jk} on O_{jh} (γ_{jkh})		
	$h = 1$	$h = 2$	$h = 3$
$j = 1$			
$k = 1$	0	0	0
$k = 2$	1	0	0
$k = 3$	0	1	0
$j = 2$			
$k = 1$	0	0	0
$k = 2$	1	0	0
$k = 3$	0	1	0
$j = 3$			
$k = 1$	0	0	0
$k = 2$	1	0	0
$k = 3$	0	1	0

A small scheduling problem instance is considered to validate the proposed model. Consequently, the validation of the model is shown by examining the result of solving the proposed optimization problem by hand and by using an optimization engine. The results are compared and are shown to be the same.

For comparative purposes, two potential solutions for the simple scheduling problem instance are considered. Each solution represents a different allocation of resources. Figure 3.3 illustrates a Gantt chart for the first potential solution.

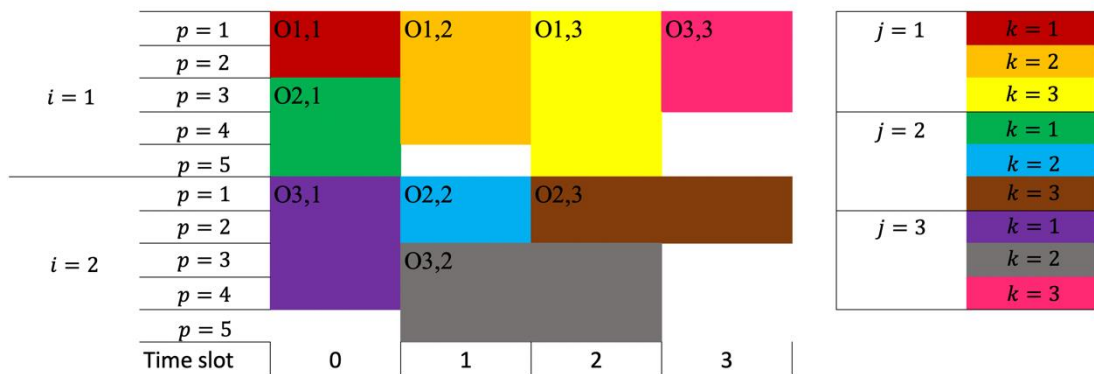


Figure 3.3: Gantt chart shows potential solution 1

In Figure 3.3, the allocation satisfies all the constraints. This can be proved by hand calculations, for more details refer to Appendix A. By substituting in the objective function, the average makespan is calculated to be 2.444 ms. Calculation details are also provided in Appendix A. The average makespan can be reduced by considering different allocations. Figure 3.4 presents a Gantt chart for another potential solution for the considered scheduling problem instance.

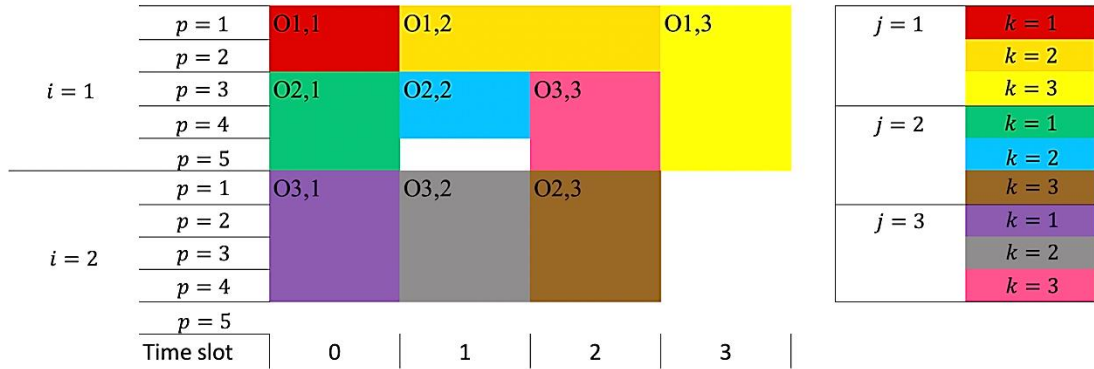


Figure 3.4: Gantt chart shows potential solution 2

By substituting in the objective function, the average makespan is found to be 2.222 ms for the second solution. Since the average makespan of the second solution is less than the average makespan of the first solution, the second solution is chosen according to the objective function. It can be proven that the second solution is optimal as follows: by the definition of time slots and compute blocks, each compute block can execute ΔP instructions per time slot. In the considered scheduling problem instance, the total instruction count for all requests is $(31 * \Delta P)$ and the total number of compute blocks is 10. Therefore, the minimum number of time slots needed to execute all the requests is 4. This is true even if there are no dependencies among operations. Since there are 3 requests only one request will be executed by the end of the time slot $t = 3$, in the best-case scenario. This gives an average makespan of 2.222 ms, as the optimal objective value. This matches with the average makespan of the second potential solution. Accordingly, the second potential solution is an optimal solution.

To claim the validation of our model, the optimization problem is solved by using the optimization engine provided by CPLEX optimization tool [38]. Moreover, the environment settings, system parameters, and requests information are entered in

the optimization tool. After running the simulation with the aforementioned setup. CPLEX provided an optimal solution that is presented in a Gantt chart in Figure 3.5.

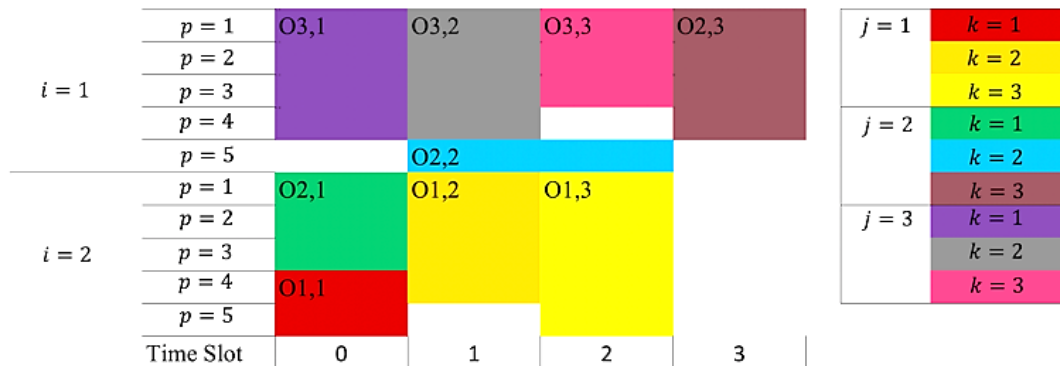


Figure 3.5: Gantt chart shows an optimal allocation of all operations by CPLEX

The solution provided by CPLEX respects all the constraints, presented in equations (10-15). Furthermore, requests $j = 1$ and $j = 3$ finished at time $t = 2$ and request $j = 2$ finished at time $t = 3$. Accordingly, given $t_d = 0.6667$ ms, the average makespan is calculated to be 2.222 ms which matches the average makespan acquired by solving the problem by hand. Accordingly, this confirms the validity of the proposed model.

Chapter 4. Decentralized Approach

In Chapter 3, the scheduling problem of allocating service requests across small cells in a 5G network is modeled as an optimization problem. In this chapter, the environment characteristics and all assumptions considering the decentralized nature of the small cells are presented. Given this decentralized environment, the interdependency problem between small cells is introduced. Furthermore, the chapter proposes a novel decentralized framework to enable the coordination between the small cells to solve the interdependency problem, consequentially leading to the global objective of minimizing the latency of fog computing in 5G networks. The proposed framework consists of multiple subproblems that are modeled and validated accordingly.

4.1. Environment Overview

The environment consists of multiple decentralized compute resources, fog units, managed by the 5G small cells. Request generating user-devices are connected to a small cell within the 5G network. A top-view centralized solution where all small cells share the scheduling knowledge to a centralized control is not adequate given the decentralization nature of the environment. By decentralization, each small cell has its own knowledge and control, meaning, it can come up with a scheduling decision.

The main challenge with the decentralized scheduling problem is that the knowledge and control are distributed among small cells. Moreover, the knowledge in each small cell is considered to be private. In reality, small cells might belong to different groups of users. For example, femtocells might be routers owned by individual users or private facilities while microcells are usually powered by an ISP. Moreover, there might be multiple small cells managed by different ISPs. Hence, in the decentralized fog computing environment in 5G, it is reasonable to assume that software agents, analogous to real world entities, are rational in the sense that they are designed and owned by people or organizations which are rational. Hereinafter, small cells are referred to as agents. The terms are used interchangeably to present the model in this chapter. Furthermore, requests are produced by user-devices that are connected to different small cell agents. Accordingly, each agent shall get a number of requests to be executed.

The major challenge with this decentralized environment is the interdependency problem among agents having no global knowledge and control. Interdependency is a goal-relevant interrelationship between actions taken by various agents [14]. In this work, the interdependency problem exists as follows:

- In the physical setup of the environment, where compute resources within each small cell agent are shared between more than one agent.
- In the mental domain of each small cell agent, where each agent has specific *capabilities* and *knowledge* about the environment.

An agent's *capability* consists of actions and control. This is represented by a set of requests, and the state of the resources on a specific instance of time. Actions are viewed as a set of operations in a request that can be executed on the compute resources within the fog computing environment at which the state of the overall environment is affected. Control is treated as a set of decision points at which the knowledge is required to select the next action to achieve the agent's goal. Control is considered to have a body of propositions representing the decision points and a state across the time dimension at which the selection process occurs. Moreover, the control can be classified as local and coordinated. Local control is the set of decision points that determine the appropriate local domain actions. Coordinated control represents the decision points that deal with the influence of the other agents on the local control.

Knowledge is treated as a set of propositions that are available if present explicitly or implicitly in the agent's memory. In this work, it refers to the knowledge required to deal with the interdependencies in a way that satisfies the local and global objectives, where the objectives are described in terms of agent interests.

Coordination between agents in such a decentralized environment can be defined as a class of solutions that provide structure and mechanism to the system to deal with the interdependency problem. Structure refers to the agent's pattern of communication and decision-making that are related to coordination. Mechanism is a composition of decision points coordinated control and interaction structure directed to resolve problems associated with interdependencies. The use of economic methods involves a process of negotiation or interaction between agents, analogous to various market behaviors. Hence, in this chapter, an economic-based method to model the

decentralized scheduling problem is proposed. Furthermore, an economic-based coordination framework to solve the interdependency problem by providing a way of collaboration between entities is also proposed. Collaboration defines the characteristic of the entities' behavior.

One of the challenges in this decentralized environment is self-interested agents. The solution to this challenge is to provide incentives to the agents in the environment that encourage individuals to behave in a way that a certain outcome prevails. In economics, an incentive is a motivational force that stimulates economic agents to a greater activity or increased efficiency. In incentive theory, there are two types of incentives: extrinsic and intrinsic incentives. Extrinsic incentive comes from an external source outside of individual agents. It can be represented as a set of rules that impose rewards or penalties by which the individualistic interests are shifted towards a common goal. Intrinsic motivation is a behavior that is resulted from internal interests rather than external influences. In this work, both extrinsic and intrinsic incentives are considered in the proposed coordination structure. In the context of 5G networks, small cells are characterized by limited resources. If the agents' local capabilities are insufficient to execute a request, collaboration would provide agents with additional resources from other agents to satisfy the request requirements. For each agent, the additional provided resources from other agents are considered as a reward, extrinsic incentive, for the collaboration represented in sharing local resources. Intrinsic motivation can be represented as small cell local interests. It is assumed that ISPs seek to provide the best quality of service. Therefore, agents aim to minimize *idle costs* of resources and the makespan of service requests. Idle cost is the opportunity cost resulted from the underutilization of resources. Collaboration enables agents to utilize their resources by sharing them with other agents, and to minimize the makespan of service requests by employing shareable available resources from other agents. In this work, it is assumed that the agents are cooperative by having the will to share their capabilities with other agents to achieve a common goal in minimizing the makespan.

Given the decentralized nature of the environment, the scheduling problem has the following assumptions to conform to the system's characteristics and environment settings:

- Small cells execute critical service requests only. Accordingly, all non-critical requests are forwarded to the cloud.
- Any user device can be connected to only one small cell at a time.
- No dependencies among requests. However, each request consists of multiple dependent or independent operations.
- All operations are non-preemptable.
- Each small cell is considered an economically rational entity with selfish behavior and has knowledge of only the local requests and resources.
- C-RAN has no knowledge of agents' private information. It does not enforce decisions but rather provides a coordination mechanism that induces agents' cooperation.

4.2. Proposed Decentralized Framework.

In this section, a novel decentralized framework is proposed to minimize the latency of fog computing in a 5G network. As discussed in the environment overview in Section 4.1, coordination between agents is the solution for the decentralized scheduling problem. This work uses economic methods which involve a process of interaction between agents, analogous to various market behaviors. An economic-based approach provides coordination between agents to solve the interdependency problem and provide a way of collaboration between agents. Hence, the following elements are extracted to build the proposed solution framework:

- *Interaction structure*: which describes the pattern of communication and decision-making that are related to coordination.
- *Self-classification*: each small cell agent as it receives a request within the 5G network determines its own type as a consumer or a provider. In the case where local resources are available, the self-classification of the agent classifies its type as a provider and the local scheduling derives the decision to execute the request based on its local knowledge and the request requirements. On the other hand, if there are no available resources that meet the request requirement, the self-classification of the agent classifies its type as a consumer, in which the requests are to be handed over to other small cell providers to execute the request.

- *Local scheduling*: handles the scheduling problem of allocating requests as they are received within the small cell to local resources with the objective of minimizing the makespan. The allocation is based on the availability, local knowledge, of compute resources within the small cell.
- *Preference representation*: each agent type has different local interests. Agents' interests are defined as preferences. An economic concept, utility function, is used to represent agent preferences. The concept of utility was used in the literature [28], [39] to value and quantify agent preferences.
- *The winner determination*: The winner determination problem is concerned with the coordinated control which represents the decision points that deal with allocating the shareable available resources of provider agents to the handed-over requests of consumer agents based on agent local preferences.

Interaction structure refers to the agent's pattern of communication and decision-making that are related to coordination. To enable the coordination among the participant agents, an interaction structure must be imposed. An interaction protocol that is based on the Contract Net Protocol [40] is utilized. A diagram showing the interaction structure between agents is depicted in Figure 4.1.

In our proposed framework, the interaction structure can be summarized in the following steps:

1. *Call for proposals (CFP)*: the coordinator signals all agents to start their scheduling process and send their proposals accordingly.
2. *Self-Classification*: each small cell classifies its own type as a consumer or provider based on its local knowledge of resources availability and request requirements.
3. *Request Classification (only for consumers)*: after the self-classification process, consumer agents classify requests into two types: local requests and handed-over requests. Local requests are to be processed by local resources while handed-over requests are to be executed by other provider agents.
4. *Local Scheduling*: requests which are classified as local in step 3, are locally scheduled based on the local resource availabilities and request requirements with the objective of minimizing the makespan.

In the following sections, the self-classification, local scheduling, preference calculation and winner determination problems are mathematically modeled as optimization problems. Furthermore, all models, as well as their objectives and constraints, are presented and discussed in detail. Finally, each proposed model is validated by hand calculations and by using an optimization engine.

4.3. Self-Classification

The self-classification problem is concerned with classifying the agent's type as a consumer or a provider. Moreover, it is related to identifying the handed-over requests that are to be executed by shared available resources. In this section, the self-classification problem is modeled as an optimization problem. The model validation is also presented.

4.3.1. Problem Formulation. The decision involved in the self-classification process depends on three main factors: the deadlines, makespan, and number of deadline-misses of requests. Formally, a deadline-miss of a request j is defined as follows:

$$d_j = \begin{cases} 1, & C_j > \tau_j \\ 0, & \text{otherwise} \end{cases} \quad (17)$$

where C_j is the makespan of the request j , defined in equation (9), and τ_j is the request deadline. Basically, $d_j = 1$ if the request is completed after its deadline, otherwise $d_j = 0$. As a result, we propose an optimization model for the self-classification problem, presented in equations (18-22):

$$\min_{x_{jkpt}} \left(\sum_j |\tau_j - C_j| * \sum_j (d_j + 1) \right) \quad (18)$$

s.t.

$$\sum_j \sum_k x_{jkpt} \leq 1, \quad \forall p, \forall t \quad (19)$$

$$x_{jkpt} \leq |(\gamma_{jkh} * |E_{jht} - 1|) - 1|, \quad \forall j, \forall k, \forall h, \forall t \quad (20)$$

$$|x_{jkp,t} - x_{jkp,t+1}| = |E_{jk,t} - E_{jk,t+1}|, \quad t_{jk}^s \leq t < T, \forall j, \forall k, \forall p \quad (21)$$

$$\sum_p \sum_t x_{jkpt} \geq \frac{P_{jk}}{\Delta P}, \quad \forall j, \forall k \quad (22)$$

The optimization problem, given in equations (18-22), is a Mixed-Integer Quadratic Programming (MIQP) problem. The decision variable x_{jkpt} represents the allocation of compute block p and time slot t to operation k of request j . The optimization objective presented in equation (18) is subject to constraints presented in equations (19-22). Constraint (19) indicates that a compute block can be allocated only to one operation at a time. Equation (20) represents the dependency constraint. Constraint (21) ensures that compute blocks allocated to an operation are not released until all the operation's instructions are executed, which represents the non-preemption assumption. Lastly, constraint (22) guarantees the execution of all operations.

According to the objective function (18), each agent schedules requests locally with the objective of minimizing the number of deadline misses. Afterward, if there are no requests pass their deadlines, then the agent classifies its own type as a provider. Otherwise, requests that pass their deadlines are considered to be handed over and the agent shall classify its own type as a consumer. To clarify the idea, consider the case of all requests having the same deadline τ_j . There are two possible scenarios for each request that affect the objective value: the request is allocated either before or after the deadline. The two scenarios are illustrated in Figure 4.2.

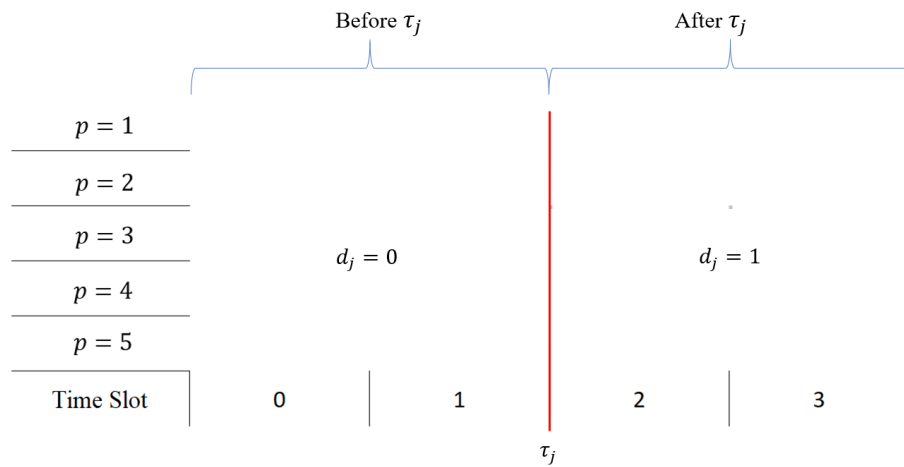


Figure 4.2: The value of d_j before and after the deadline

Formally, a request j is a to be handed over if $d_j = 1$ or locally scheduled if $d_j = 0$. Moreover, an agent type is classified as a provider if $\sum_j^J d_j = 0$, otherwise, the agent type is classified as a consumer. In the next subsection, the validation of the proposed self-classification model is presented.

4.3.2. Model Validation. Some parameters in this setup need to be set in order to validate and solve the problem and acquire an optimal solution. Some of the parameters reflect the environment such as the number of agents and the number of requests and their operations. These parameters might influence the type of each agent. Other parameters are reflecting the settings that can be modified by the system administrator, i.e. ISP, such as the number of compute blocks and their capabilities which also affect the duration of each time slot. Table 4.1 summarizes all system parameters considered in the validation process of all models.

Table 4.1: Summary of the system parameter values for the decentralized models

Parameter	Value
Agent CPU speed	1 GHz
Agent Performance	1.3 Cycle Per Instruction
CB CPU speed (Cycles)	200 MHz
CB CPU speed (ins/sec)	Approx. 150 MIPS
Duration of each time slot	666.7 ms
CB CPU speed (Instruction/time slot)	0.1 MIPS
Time Window	10 time slots

A small problem instance that involves two agents, a provider and a consumer, is used to test the system behavior. Each agent is characterized by a number of compute blocks and a number of local requests and their operations. The information regarding the two considered agents is provided in Table 4.2.

Some dependencies exist among operations of some requests. Dependencies are presented in a matrix form in Chapter 3. Similarly, the dependency information of each operation k of each task j of the considered problem instance is shown in Table 4.3.

To validate the model, the self-classification constraints are similar to the constraints validated in our proposed scheduling model in Section 3.7. Hence, the focus is on the objective function to validate system behavior. Agent $i = 1$ has two requests, each has a deadline of $\tau = 6$. A potential solution for the self-classification problem, for agent $i = 1$, is represented in Figure 4.3.

Table 4.2: Agents and requests information 1

Agent i	Num. of CBs	Request j	Instruction count ($\times 10^5$) IPS				Deadline (time slot)
			For operation O_{jk}				
			$k = 1$	$k = 2$	$k = 3$	$k = 4$	
$i = 1$	6	$j = 1$	2	3	4	5	$\tau_1 = 6$
		$j = 2$	2	3	4	3	$\tau_2 = 6$
$i = 2$	5	$j = 1$	2	2	4	5	$\tau_1 = 6$
		$j = 2$	2	3	4	5	$\tau_2 = 7$
		$j = 3$	2	3	4	3	$\tau_3 = 9$
		$j = 4$	2	1	2	3	$\tau_4 = 8$
		$j = 5$	2	3	4	----	$\tau_5 = 6$

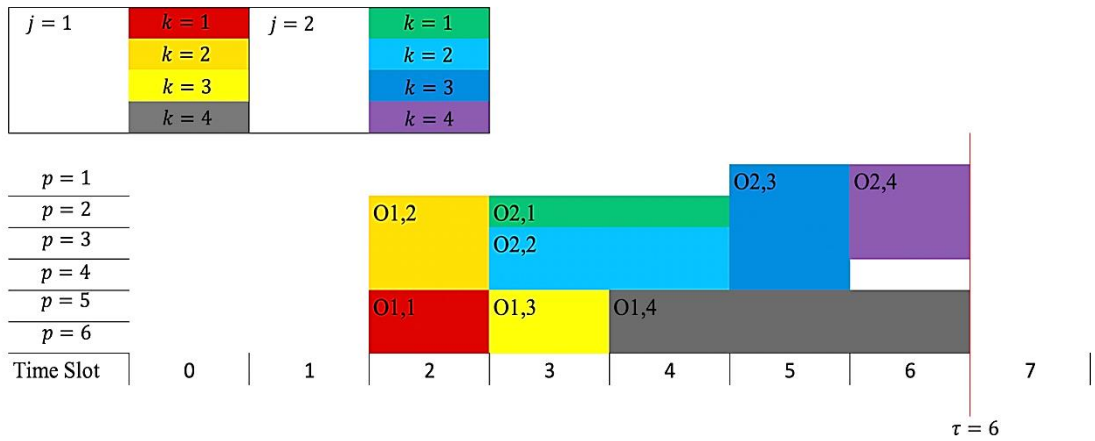


Figure 4.3: Gantt chart shows a potential solution of self-classification for agent $i = 1$

The allocation in Figure 4.3 satisfies all the model constraints. This can be proven by a hand, similar to what is presented in Appendix A. Afterward, by substituting into the objective function, equation (18), the objective value is calculated as “0”. The calculation steps are provided in Appendix B. This value means that all

requests will be locally scheduled and executed, i.e. $\sum_j^l d_j = 0$. As a result, agent $i = 1$ is classified as a provider agent.

Table 4.3: Dependency information 2

Agent i	Dependency of O_{jk} on O_{jh} (γ_{jkh})						
$i = 1$	$j = 1$	$h = 1$	$h = 2$	$h = 3$	$h = 3$	$h = 4$	
	$k = 1$	0	0	0	0	0	
	$k = 2$	0	0	0	0	0	
	$k = 3$	1	1	0	0	0	
	$k = 4$	0	0	1	0	0	
	$j = 2$	$h = 1$	$h = 2$	$h = 3$	$h = 3$	$h = 4$	
	$k = 1$	0	0	0	0	0	
	$k = 2$	0	0	0	0	0	
	$k = 3$	0	0	0	0	0	
	$k = 4$	0	0	0	0	0	
	$i = 2$	$j = 1$	$h = 1$	$h = 2$	$h = 3$	$h = 3$	$h = 4$
		$k = 1$	0	0	0	0	0
		$k = 2$	0	0	0	0	0
		$k = 3$	1	1	0	0	0
$k = 4$		0	0	1	0	0	
$j = 2$		$h = 1$	$h = 2$	$h = 3$	$h = 3$	$h = 4$	
$k = 1$		0	0	0	0	0	
$k = 2$		0	0	0	0	0	
$k = 3$		1	1	0	0	0	
$k = 4$		0	0	1	0	0	
$j = 3$		$h = 1$	$h = 2$	$h = 3$	$h = 3$	$h = 4$	
$k = 1$		0	0	0	0	0	
$k = 2$		0	0	0	0	0	
$k = 3$		1	1	0	0	0	
$k = 4$		0	0	1	0	0	
$j = 4$		$h = 1$	$h = 2$	$h = 3$	$h = 3$	$h = 4$	
$k = 1$		0	0	0	0	0	
$k = 2$		0	0	0	0	0	
$k = 3$		1	1	0	0	0	
$k = 4$		0	0	1	0	0	
$j = 5$		$h = 1$	$h = 2$	$h = 3$	$h = 3$	$h = 4$	
$k = 1$		0	0	0	0	---	
$k = 2$		1	0	0	0	---	
$k = 3$		0	1	0	0	---	
$k = 4$		---	---	---	---	---	

Similarly, for agent $i = 2$, a possible solution can be illustrated in a Gantt chart represented in Figure 4.4. In Figure 4.4, all the compute blocks are allocated to requests.

Therefore, there is no other resource allocation that results in a better objective value without violating any constraint. For example, $O_{3,3}$ and $O_{3,4}$ cannot be executed earlier due to the operations' resource and dependency requirements. By substituting into the objective function in (18), the objective value is calculated as “4”. Since $\sum_j^J d_j = 1$, this agent is classified as a consumer agent. Accordingly, request $j = 3$ is considered to be handed over.

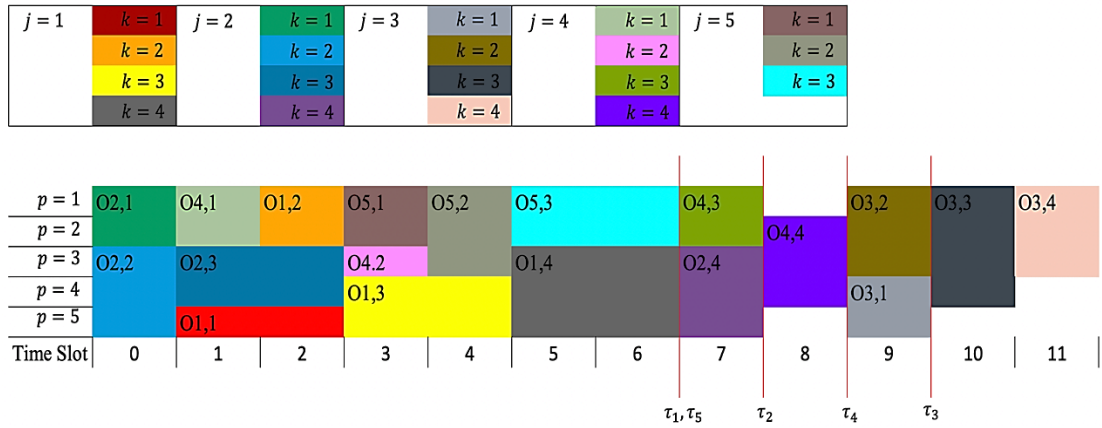


Figure 4.4: Gantt chart shows a potential solution of self-classification for agent $i = 2$

For validation purposes, the model is implemented in CPLEX optimization tool. The environment settings, system parameters, and requests information are entered in the optimization tool. After running the optimization engine, the optimal objective values for the self-classification of the provider and the consumer are “0” and “4”, respectively. This confirms the expected behavior of the system and the hand calculations. Accordingly, this shows the validity of the proposed simple model instance.

4.4. Local Scheduling

In Section 4.3, the proposed self-classification model was introduced. Following the proposed framework presented in Section 4.2, after identifying the requests to be handed over in the process of the self-classification, each agent starts the local scheduling process of the remaining requests. In this section, an optimization model is proposed that defines the decision-making involved in the local scheduling process.

4.4.1. Problem Formulation. The decision-making of the local scheduling process is based on the local knowledge of the resource availabilities and request requirements. Therefore, the local scheduling problem is defined based on the scheduling model (9-15) in Section 3.7 as follows:

$$\min_{x_{jkpt}} \sum_j^J C_j \quad (23)$$

s.t.

$$C_j + D_T(j) + D_p \leq \tau_j, \quad \forall j \quad (24)$$

$$\sum_j^J \sum_k^K x_{jkpt} \leq 1, \quad \forall p, \forall t \quad (19)$$

$$x_{jkpt} \leq |(Y_{jkh} * |E_{jht} - 1|) - 1|, \quad \forall j, \forall k, \forall h, \forall t \quad (20)$$

$$|x_{jkp,t} - x_{jkp,t+1}| = |E_{jk,t} - E_{jk,t+1}|, \quad t_{jk}^s \leq t < T, \forall j, \forall k, \forall p \quad (21)$$

$$\sum_p^P \sum_t^T x_{jkpt} \geq \frac{P_{jk}}{\Delta P} \quad (22)$$

The optimization model represents an MIQCP problem that aims to schedule all requests with the objective of minimizing the makespan. The objective function presented in equation (23) is subject to constraints (19-22) and (24). Constraints (19-22) are explained in Section 4.3 while constraint (24) is the deadline constraint. The following subsection presents the validation of the proposed local scheduling model.

4.4.2. Model Validation. To validate the local scheduling model, the example previously provided to validate the self-classification model, given in Table 4.2, is used to show the flow of the decentralized scheduling process throughout the proposed framework. Accordingly, the validation of the model is shown by examining the result of solving the proposed local scheduling optimization problem by hand and by using an optimization engine. The results are compared and are shown to be the same. A possible solution for agent $i = 1$ is presented in Figure 4.5.

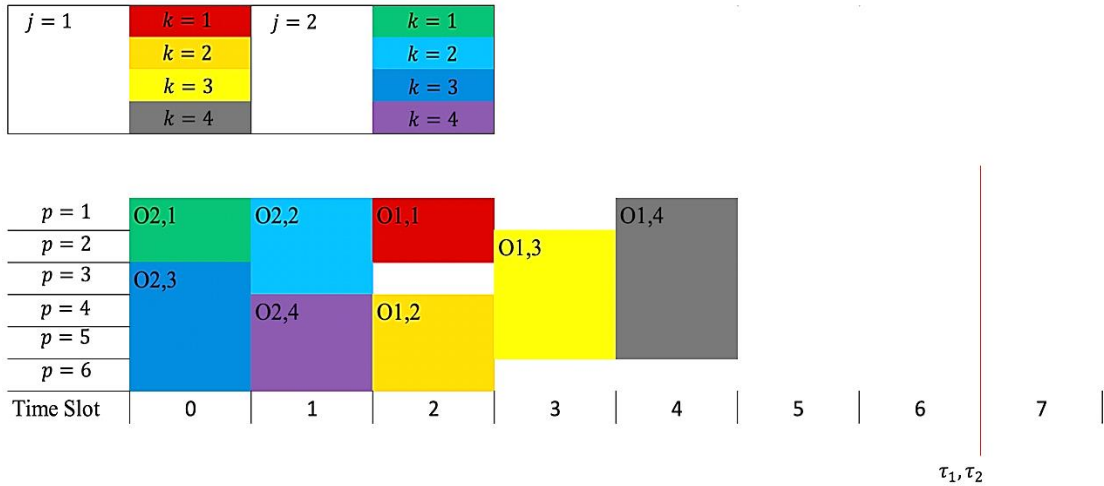


Figure 4.5: Gantt chart shows an example of local scheduling for agent $i = 1$

In Figure 4.5, operations $O_{1,4}$ and $O_{2,4}$ finish at time slots $t = 1$ and $t = 4$, respectively. By substituting into the objective function (23), the objective value is obtained to be “7”. Steps are provided in Appendix B. The same process can be applied to the example of agent $i = 2$, the consumer agent.

In the process of validating the proposed local scheduling model, the CPLEX optimization tool is used to obtain an optimal solution for the considered small local scheduling problem instance. The objective value of the schedule resulted from the CPLEX simulation is “7”. Accordingly, the proposed local scheduling model is valid since the objective value calculated by hand is identical to the objective value obtained from the optimization engine.

To continue with the scheduling process conforming to the proposed framework, the local scheduling problem instance of the consumer agent $i = 2$ is solved using the CPLEX optimization tool. The resultant schedule can be shown in the Gantt chart illustrated in Figure 4.6.

In Figure 4.6, the objective value acquired from CPLEX is “25”. Furthermore, the request $j = 3$ in agent $i = 2$, that is identified as a handed-over request by the self-classification model in subsection 4.3.2, is not present in the obtained local scheduling result. This conforms with the expected system behavior.

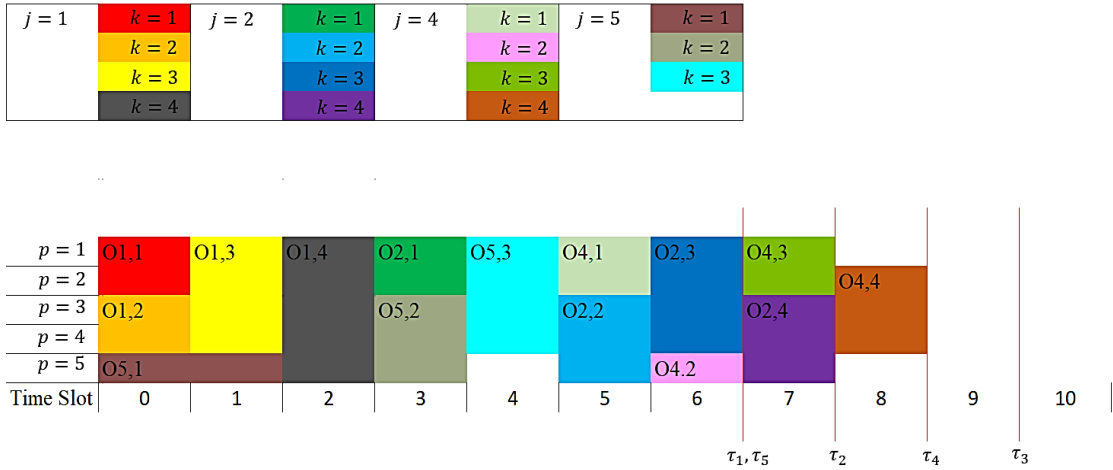


Figure 4.6: Gantt chart shows CPLEX local scheduling for agent $i = 2$

4.5. Winner Determination

After the self-classification and local scheduling processes, the handed-over requests are executed by shareable available resources. Accordingly, agents need to collaborate to overcome such an interdependency and achieve the global objective of minimizing the makespan. As part of the coordination mechanism proposed in Section 4.2, the decision-making of which the shareable available resources are allocated to the handed-over requests based on the agents' preferences is known in the literature as the winner determination problem. In this section, the winner determination problem is modeled as a combinatorial auction problem using a game-theoretic approach.

4.5.1. Problem Formulation. Let the winner determination problem be considered as a game. In addition, let all the agents involved in the problem be the players in this game. Accordingly, there are two types of players, providers, and consumers. Moreover, the goal for each player is to maximize its own payoff which reflects its self-interest. A player's interest depends on its type. A provider's interest can be related to its resources' utilization. On the other hand, a consumer's preference can be related to its requests' makespan. Additionally, agents shall have their own strategies to maximize their payoffs. Furthermore, game rules must be enforced to lead to a global solution in which no other solutions can minimize the latency further. Moreover, this game is concerned with scheduling all tasks to all compute blocks located within each agent. Such individualistic details are further provided by the non-cooperative game theory. Besides, requests are to be generated continuously, so the

game is to be repeated infinitely. Also, 5G networks are characterized to be dynamic in nature which makes taking decisions based on historic data unreliable. Therefore, players shall choose their strategies independently of other players' previous decisions. Hence, our game can be considered to be an effectively simultaneous game. Additionally, all agents are assumed to behave the same, which means that player strategies are considered as common knowledge. However, players have no knowledge regarding each other local requests or available resources. Consequently, this game has complete but imperfect information. Moreover, the number of possible actions by all players can be extremely huge due to the variation of items in interest. In specific, agents are after combinations of compute and time resources. Therefore, the possibility of having innumerable combinations of both classifies the game as combinatorial. To summarize, the game describing the problem is a non-cooperative simultaneous combinatorial game with complete but imperfect information.

Auction-based modeling has been used in the literature to represent games [39]. Therefore, the game under study can be modeled as an auction where provider players are the sellers and consumer players are the buyers. Furthermore, player payoffs can be represented as agent utilities. Also, provider strategies can be expressed as the reserved price of their goods while consumer strategies can be represented as their willingness-to-pay. The auctioneer's objective is to maximize the sum of all utilities. Additionally, resource blocks are goods offered by the sellers and purchased by the buyers. A resource block is an allocation of a compute block in a time slot. Moreover, resource blocks are offered in the auction in various combinations with different numbers of compute blocks and time slots. Let each possible combination be $a \in \{1, \dots, A\}$ where A is the number of offered goods, combinations of shareable available resources. Therefore, the winner determination problem can be represented as a combinatorial auction problem as follows:

$$\max_{x_{jia}} \sum_j^J \sum_i^I U_j(a) * x_{jia} \quad (25)$$

s.t.

$$\sum_j^J (r_{apt}(j) * x_{jia}) + Q_{ipt} \leq 1, \quad \forall i, \forall p, \forall t, \forall a \quad (26)$$

$$\sum_i^I x_{jia} \leq 1, \quad \forall j, \forall a \quad (27)$$

The combinatorial auction problem is a Mixed Integer Linear Programming (MILP) optimization problem. The objective function (25) aims to maximize the utility function $U_j(a)$. In General, the utility value reflects agent preferences, willingness-to-pay and reserved values, given that combination a of resources will execute request j . Therefore, the utility function reflects the buyer preferences for buying compute resources, and the reserve price for combinations of goods offered by the sellers. In the objective function (25), x_{jia} is a decision variable, where $x_{jia} = 1$ if a buyer wins a combination a of resources from seller i to execute a request j . In constraint (26), $r_{apt}(j)$ represents the requirement for request j of possible allocations of compute resources. The variable Q_{ipt} represents the availability of shareable compute resources (p, t) . In other words, $Q_{ipt} = 0$ if compute block p in agent i at time slot t is available, and $Q_{ipt} = 1$ otherwise. Moreover, constraint (26) prohibits selling a resource block to multiple buyers. Accordingly, it represents stock availabilities. Finally, constraint (27) enforces any sold combination to be offered by the same seller.

Moreover, sealed-bid first-price auction is used since the game is a simultaneous game with complete but imperfect information. Therefore, each request's requirement $r_{apt}(j)$, and its utility $U_j(a)$ are calculated by its corresponding buyer agent independently of good availabilities in the market. The calculated utilities are referred to as bids. Then calculated bids are submitted to the auctioneer in order to determine the winner. In the next sections, the calculation and representation of agent preferences and the corresponding utility values are presented.

4.5.2. Agent preferences. In auctions, each buyer shall bid for goods depending on local preferences that define their willingness-to-pay. Additionally, each buyer shall submit a bid for each request that demands shareable available resources. Likewise, each seller shall value its offered goods, resource blocks, that depend on local preferences, expressed as reserved values. A fixed price for each resource block of one currency unit is assumed. On the other hand, buyer preferences are based on the makespan of their requests.

For buyers to calculate their utilities, some information about the market should be shared. Let each unique capability be $\{g_n \mid 0 \leq n \leq G\}$ where G is the number of unique capabilities. Also, we assume $g_0 = 0$ as a dummy value for modeling purposes. For example, if the market has 4 sellers with capabilities of 5, 6, 5 and 2, then $g_n \in \{0, 5, 6, 2\}$ and $G = 3$. Accordingly, the auctioneer sends the set of unique capabilities to all buyers. However, buyer agents have no information about the available resources offered in the market. Therefore, buyer preferences shall be calculated by considering all possible combinations of available resource blocks independently of the shareable resources' availability. More specifically, buyers lack the knowledge of sellers' available resources. Hence, all subsets of available resource blocks and their different combinations shall be assumed and be considered in calculating the preferences. Let each possible combination be $a \in \{1, \dots, A\}$, where $A = \sum_n^G (2^{g_n * T} - 1)$ is the number of all possible combinations of available resources. Moreover, each combination a of available resource blocks can be represented in a binary function $M_{pt}(a)$, such that $M_{pt}(a) = 0$ only if a compute block p in a resource combination a at time slot t is assumed available, and $M_{pt}(a) = 1$ otherwise, where $1 \leq p \leq P_a$, $1 \leq a \leq A$, $0 \leq t < T$ and $P_a = \{g_n \mid 2^{g_{n-1} * T} \leq a < 2^{g_n * T}\}$ is the capability considered in a combination a of available resource blocks. For further understanding, consider one seller agent, $G = 1$, with a capability of 2 compute blocks, $g_1 = 2$. In this example, the time window is assumed to be two time slots, $T = 2$. All subsets of available resource blocks and their different possible combinations are shown in Figure 4.7.

In Figure 4.7, the number of all possible combinations is $A = \sum_n^1 (2^{g_n * 2} - 1) = 2^{2*2} - 1 = 15$. In this example, the values of $M_{pt}(a)$ for the possible available resource combination $a = 2$ are $M_{1,0}(2) = 1$, $M_{1,1}(2) = 0$, $M_{2,0}(2) = 0$ and $M_{2,1}(2) = 0$. Similarly, the values of $M_{pt}(a)$ for the combination $a = 13$ are $M_{1,0}(13) = 1$, $M_{1,1}(13) = 0$, $M_{2,0}(13) = 1$ and $M_{2,1}(13) = 1$. Lastly, the values of $M_{pt}(a)$ considering the first combination $a = 1$ are all zeros since all resource blocks are assumed to be available in this resource combination.

The preference of each buyer is calculated based on the makespan of the request for which the buyer is submitting a bid. In addition, the makespan calculation depends on the shareable available resources. However, since buyer agents lack the knowledge

of the shareable resources' availability, all possible available resource combinations shall be considered. Therefore, buyers shall calculate their preferences for every possible combination $a \in \{1, \dots, A\}$.

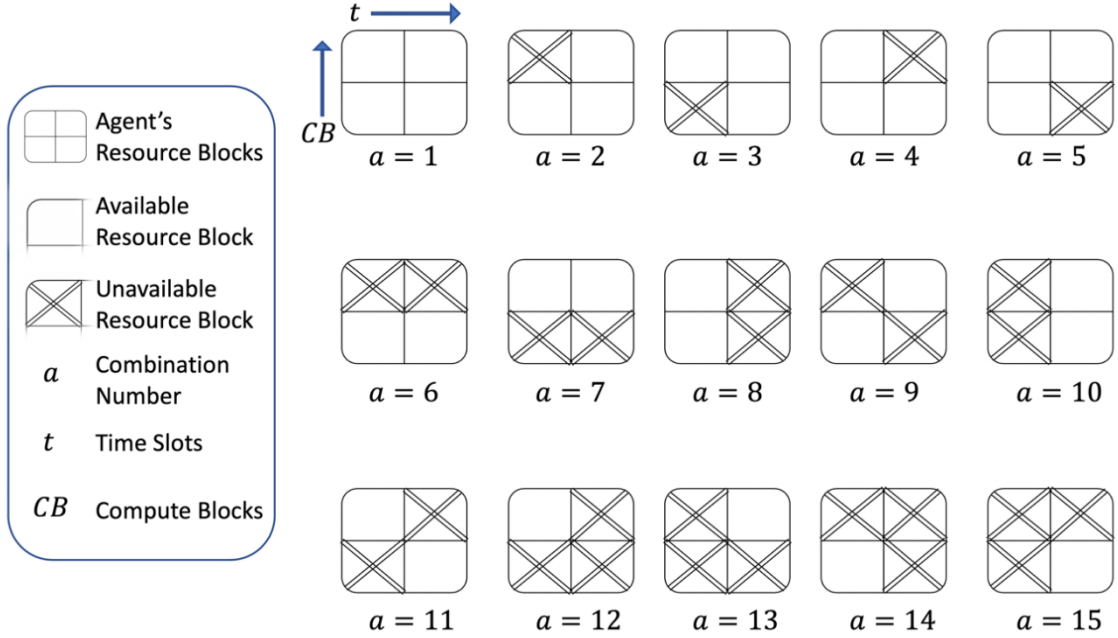


Figure 4.7: All combinations of available resource blocks (Example)

To calculate the makespan of a request j assuming a combination a of available resources, the buyer agent needs to schedule the request with the objective of minimizing the makespan. As a result, the schedule and the corresponding minimum makespan can be acquired by solving the following optimization problem:

$$C_j^{min}(a) = \min_{x_{jkapt}} \max_k \left(t_{jk}^s + \text{ceil} \left(\frac{P_{jk}}{\sum_p^{Pa} x_{jkapt} * \Delta P * t_d} \right) \right), \quad \forall j, \forall a \quad (28)$$

s.t.

$$\sum_k^K x_{jkapt} + M_{pt}(a) \leq 1, \quad \forall j, \forall a, \forall p, \forall t \quad (29)$$

$$x_{jkapt} \leq |(\gamma_{jkh} * |E_{jht} - 1|) - 1|, \quad \forall j, \forall a, \forall k, \forall h, \forall t \quad (30)$$

$$C_j^{min}(a) + D_T(j) + D_p < \tau_j, \quad \forall j, \forall a \quad (31)$$

$$|x_{jkap,t} - x_{jkap,t+1}| = |E_{jk,t} - E_{jk,t+1}|, \quad t_{jk}^s \leq t < T, \forall j, \forall k, \forall a, \forall p \quad (32)$$

$$\sum_p^{P_a} \sum_t^T x_{jkapt} \geq \frac{P_{jk}}{\Delta P}, \quad \forall j, \forall k, \forall a \quad (33)$$

Similar to the optimization problem presented in (9-15), the makespan calculation represented in equations (28-33) is an MIQCP optimization problem. The variable x_{jkapt} is a decision variable where $x_{jkapt} = 1$ if a compute block p in a time slot t in a given combination a is allocated to an operation k of a given request j , and $x_{jkapt} = 0$ otherwise. The objective value $C_j^{min}(a)$, calculated by solving the optimization problem (28-33), is the minimum makespan possible by allocating a request j considering a combination a of available resources. The objective function in (28) is subject to constraints (29-33). Equation (29) prohibits allocating a resource block to multiple operations. In addition, it prevents allocating the presumably unavailable resources of a combination a . Also, equation (30) is the dependency constraint of a request's operations. Furthermore, equation (31) is the deadline constraint. Constraint (32) ensures the assumption of non-preemptable operations. Finally, constraint (33) guarantees the execution of all operations.

The preference of a buyer agent is defined as the difference between the minimum makespan, $C_j^{min}(a)$, and the time window, t^{max} . On the other hand, the preference of a seller is represented as the reserved price for their offered goods. Therefore, the utility value, the bidding value, should include the preferences of both agent types, sellers and consumers. The bids calculation and generation are presented in the next subsection.

4.5.3. Utility calculation and bid generation. The bidding value for a request j assuming a possible combination a of shareable available resources is calculated as follows:

$$U_j(a) = \sum_k^{K_j} \frac{P_{jk}}{\Delta P} + t^{max} - C_j^{min}(a), \quad \forall a, \forall j \quad (34)$$

The utility function $U_j(a)$ represents the willingness-to-pay of a buyer to schedule request j assuming a combination a of available resources. The first term is the number of required resource blocks to execute task j . This represents the reserved price assuming that the price for each resource block is one currency unit. The utility function is defined as the sum of all resource blocks needed plus how early the task finishes execution. The greater the number of required compute blocks, the higher the willingness of the buyer to pay to cover for the reserved price of the goods. In addition, the earlier the finishing time of a request, the lower the makespan and the greater the bidding value. Consequently, the utility function in (34) reflects the preferences of both buyer and seller agents.

After buyers calculate their utilities and define their requirements, they generate a bid $b_j = \{\rho_{ij}, U_j(a), r_{apt}(j)\}$ for each request j where ρ_{ij} is a globally unique ID of the bid, $U_j(a)$ is the calculated utilities, bidding values, and $r_{apt}(j) = \sum_k^{K_j} x_{jkapt}$, $\forall j, \forall a, \forall p, \forall t$ is the requirement for request j . All buyers shall submit their bids to the auctioneer to conduct the auction and determine the winner by solving the combinatorial auction problem defined in equation (25).

4.5.4. Model Validation. To validate the winner determination model, a simple problem instance consists of multiple providers and a consumer is considered. For the purpose of showing the whole process of the proposed framework, presented in Section 4.2, the two agent examples that were involved in the validation process of the self-classification and local scheduling models, presented in Table 4.2, are to be used to validate the winner determination model. In addition, one more example of a provider agent, namely agent $i = 3$, is considered. Table 4.4 summarizes the information regarding the three considered agents and the corresponding requests. Agent $i = 3$ has only one request for the purpose of illustrating the scenario where there is a provider with extra sharable available resources. The dependency information regarding the request $j = 1$ of agent $i = 3$ is shown in Table 4.5. Moreover, the dependency information regarding the requests of agents $i = 1$ and $i = 2$ is previously given in Table 4.3.

For agents $i = 1$ and $i = 2$, the self-classification and local scheduling processes are presented in Sections 4.3 and 4.4, respectively. Furthermore, the same

procedure of using CPLEX engine is followed for agent $i = 3$ to continue with the validation process of the winner determination model. As a result, agent $i = 3$ is classified as a provider and the local scheduling result is illustrated in the Gantt chart shown in Figure 4.8.

Table 4.4: Agents and requests information 2

Agent i	Num. of CBs	Request j	Instruction count ($\times 10^5$) IPS For operation O_{jk}				Deadline (time slot)
			$k = 1$	$k = 2$	$k = 3$	$k = 4$	
$i = 1$	6	$j = 1$	2	3	4	5	$\tau_1 = 6$
		$j = 2$	2	3	4	3	$\tau_2 = 6$
$i = 2$	5	$j = 1$	2	2	4	5	$\tau_1 = 6$
		$j = 2$	2	3	4	5	$\tau_2 = 7$
		$j = 3$	2	3	4	3	$\tau_3 = 9$
		$j = 4$	2	1	2	3	$\tau_4 = 8$
		$j = 5$	2	3	4	----	$\tau_5 = 6$
$i = 3$	6	$j = 1$	2	2	4	5	$\tau_1 = 6$

Table 4.5: Dependency information of agent 3

Agent i	Dependency of O_{jk} on O_{jh} (γ_{jkh})					
$i = 1$	$j = 1$	$h = 1$	$h = 2$	$h = 3$	$h = 3$	$h = 4$
	$k = 1$	0	0	0	0	0
	$k = 2$	0	0	0	0	0
	$k = 3$	1	1	0	0	0
	$k = 4$	0	0	1	0	0

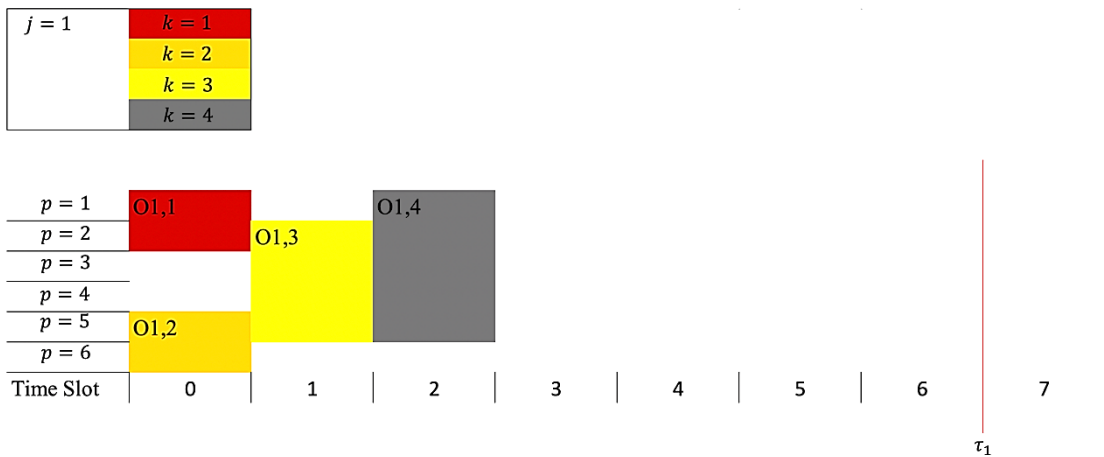


Figure 4.8: Gantt chart shows CPLEX local scheduling for agent $i = 3$

As per the results presented in the validation process of the self-classification model in Section 4.3, request $j = 3$ in agent $i = 2$ is considered to be handed over. Therefore, the handed-over request globally unique ID is defined as $j = \rho_{23}$. For the sake of this example, the globally unique ID is assumed to be $\rho_{23} = 1$. In this example, there are only two possible scenarios in which request $j = 1$ buys resources from either agent $i = 1$ or agent $i = 3$. A possible solution for each scenario is considered and the corresponding objective values are compared. Figure 4.9 depicts an example allocation in case of agent 1 resources are bought by request $j = 1$.

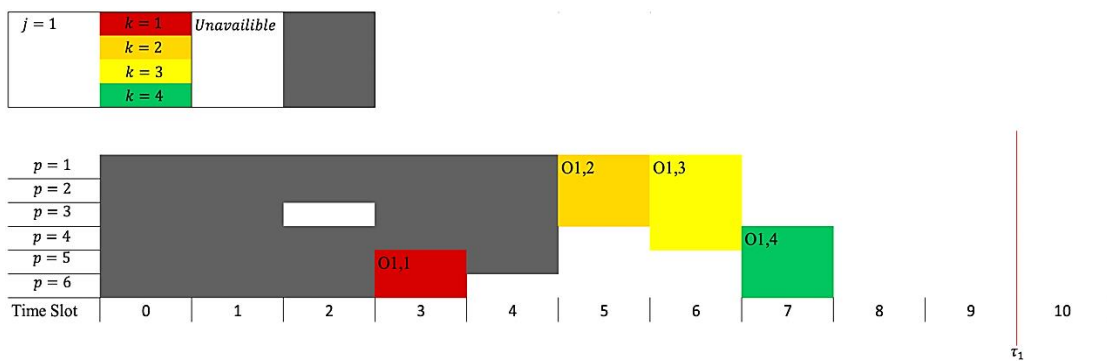


Figure 4.9: Gantt chart shows the allocation of handed-over request $j = 1$ in agent $i = 1$

In Figure 4.9, the grey shaded area represents the unavailable resource blocks in agent $i = 1$. By substituting in (34), the utility value is calculated to be 13 currency units. All the calculation steps are provided in Appendix B. Moving forward with the next scenario, the potential solution of agent $i = 3$ selling resources to request $j = 1$ is shown in Figure 4.10.

In Figure 4.10, the grey shaded area represents the unavailable resources blocks in agent $i = 3$. By substituting into eq. (34), the utility value is calculated to be 15 currency units. All the calculation steps are provided in Appendix B. According to the objective function (25), request $j = 1$ is to be executed by agent $i = 3$ since this maximizes the sum of all utilities. CPLEX simulation is performed and resulted in identical objective value and final schedule with an average makespan of 5.125 time slots. To validate the average makespan, the following equation (35) is used.

$$\text{Avg. makespan} = \frac{\text{total makespan}}{\text{number of requests}} \quad (35)$$

Accordingly,

$$\text{Avg. makespan} = \frac{7 + 25 + 3 + 6}{8} = 5.125 \text{ time slots}$$

The obtained CPLEX results and the hand calculations are identical for the provided simple example. Hence, this presents the validity of our model. Although this model describes the combinatorial auction problem successfully, the winner determination process involved is known to be an NP-hard problem. This means that there is no algorithm that can acquire the optimal solution in polynomial time. In the next chapter, the proposed auction-based heuristic to obtain an adequate solution in polynomial time is presented.

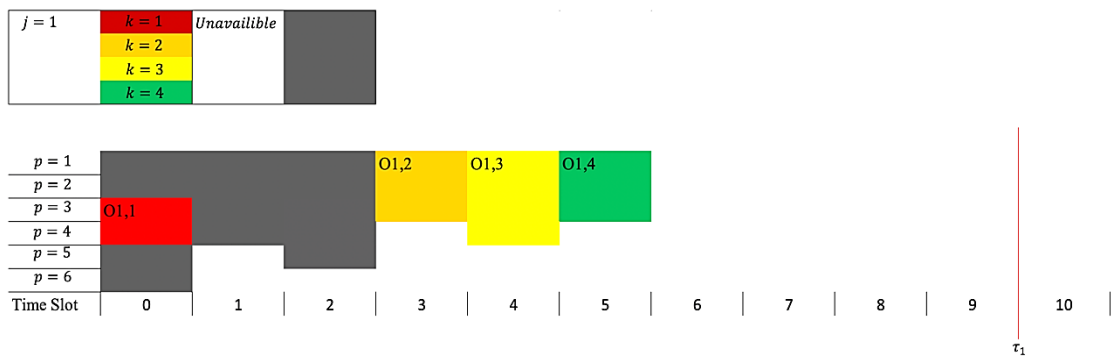


Figure 4.10: Gantt chart shows the allocation of handed-over request $j = 1$ in agent $i = 3$

Chapter 5. Decentralized Auction-Based Heuristic Solution

In Chapter 4, the interdependency problem between the decentralized small cells is introduced. Furthermore, a proposed framework is presented to solve the interdependency problem. According to the framework, the decentralized scheduling problem is divided into three subproblems: self-classification, local scheduling and winner determination problems. Each of the three subproblems is modeled as an NP-Hard optimization problem [37]. Therefore, a heuristic technique is needed to find a solution that is close to the optimal in polynomial time. Hence, a decentralized auction-based heuristic technique is proposed to solve the scheduling problem in fog computing within the 5G network. In this chapter, the proposed heuristic solution and its components are presented.

5.1 Decentralized Auction-Based Solution Overview

According to the proposed framework presented in Section 4.2, agents have the capabilities of self-classification and local scheduling. Both capabilities are modeled as NP-Hard scheduling problems. To enable the functionality of such capabilities in polynomial time, the following is proposed: each small cell agent shall schedule all its requests locally. Subsequently, each agent checks whether all requests will be executed before their deadlines. If all requests are completed before the corresponding deadlines, the scheduling agent classifies its own type as a provider. However, if any request is completed after the corresponding deadline, the request is considered to be handed over and executed by another agent. Accordingly, the scheduling agent classifies its own type as a consumer. Afterward, consumers remove the handed-over request and reschedule all the remaining requests, rechecking if all deadlines are respected. In the case of multiple handed-over requests, consumer agents remove the request smallest in size. Consumer agents keep removing handed-over requests and rescheduling the remaining requests until all deadlines are respected. Eventually, the removed requests are to be handed over while the remaining requests are scheduled locally. The local scheduling process is an NP-hard problem; therefore, a heuristic scheduling algorithm is required to obtain adequate solutions in polynomial time. Hence, a developed Simulated Annealing-Based Scheduling algorithm, denoted as (SABS), is proposed to tackle the agents' local scheduling problem.

In Chapter 4, a decentralized framework is proposed to enable the coordination between small cells to solve the interdependency problem. As per the proposed framework, the coordinated control which represents the decision points that deal with allocating the shareable available resources of provider agents to the handed-over requests of consumer agents based on agent local preferences is defined as the winner determination problem. Moreover, agents calculate utility values that reflect their local preferences as part of the problem. In Section 4.5, the winner determination problem is modeled as a combinatorial auction problem that is unsolvable in polynomial time. Therefore, an Auction-Based Winner Determination algorithm (ABWD) is proposed in this chapter to obtain an adequate solution in a time that complies with 5G standards.

By utilizing the two proposed algorithms, SABS and ABWD, the workflow of the proposed decentralized auction-based heuristic is depicted in Figure 5.1 and represented in the following steps:

1. *Signal All Agents*: the scheduling system starts with a broadcast signal from the coordinator to all agents.
2. *Local Scheduling (SABS)*: all agents start executing the proposed SABS algorithm to locally schedule the requests received by user-devices in the 5G network.
3. *Check Deadline-Misses*: each agent checks the deadline constraint of each request. If there are no deadline-misses, go to step 9. Otherwise, continue to step 4.
4. *Self-Classify as A Consumer*: the agent's type is classified as a consumer.
5. *Remove A Request*: each consumer agent removes the request that misses its deadline. In the case of multiple deadline-misses, the consumer agent removes the request smallest in size that missed its deadline.
6. *Classify the Request*: the removed request is classified as a handed-over request to be executed by another agent.
7. *Local Scheduling (SABS)*: consumers reschedule the remaining local requests using the proposed SABS algorithm.
8. *Check Deadline-Misses*: each consumer agent checks the deadline constraint of each remaining local request. If there is any deadline-miss, go to step 5. Otherwise, go to step 10.

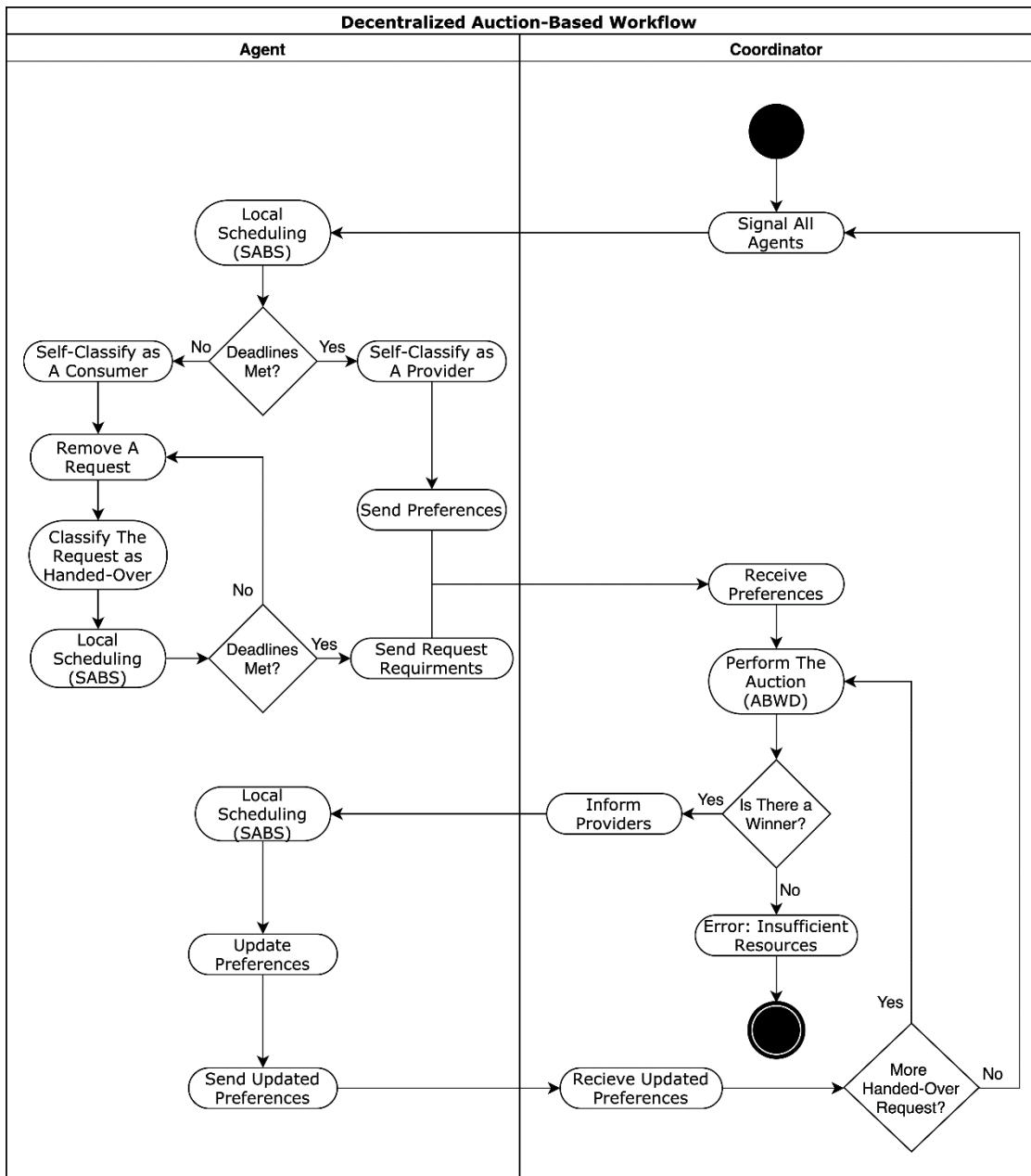


Figure 5.1: A diagram shows the proposed solution workflow

9. *Self-Classify as A Provider*: the agent's type is classified as a provider.
10. *Send Preferences*: all agents shall send their preferences to the coordinator. In specific, provider agents send the reserved values of their goods while consumer agents send the requirements of the handed-over requests.
11. *Perform the Auction (ABWD)*: after the coordinator receives all preferences, the proposed ABWD algorithm is used to solve the winner determination problem.
12. *Check the Result*: if step 11 resulted in a decision by which a handed-over request is to be executed by shareable available resources, continue to step 13.

Otherwise, terminate because there are insufficient available resources to satisfy the handed-over request requirements.

13. *Inform All Providers*: all providers receive the corresponding handed-over requests based on the results obtained by ABWD algorithm in step 11.
14. *Local Scheduling (SABS)*: each provider locally schedules the received handed-over requests using the proposed SABS algorithm.
15. *Update Preferences*: after the local scheduling process, each provider shall update its preferences based on the remaining available local resources to be shared.
16. *Send Updated Preferences*: updated provider agents' preferences are sent to the coordinator to accommodate the remaining handed-over requests if any.
17. *Check the Remaining Handed-Over Requests*: if there are remaining handed-over requests that need to be executed by provider agents, go to step 11. Otherwise, go to step 1.

In the following sections, the heuristic SABS and ABWD algorithms are presented and discussed in detail.

5.2 Simulated Annealing-Based Scheduling (SABS) Algorithm

In Chapter 3, the scheduling problem is modeled considering full knowledge of the agents and the environment settings of the 5G network. Since a top-view solution is not adequate given the decentralized nature of the environment, a decentralized approach is proposed in Chapter 4. In the proposed decentralized solution, the scheduling process is dispersed such that each agent locally schedules the requests it receives within the 5G network. Therefore, the SABS algorithm is generalized such that it provides a solution for the scheduling model, proposed in Section 3.7, and the agent's local scheduling process, presented in Section 4.4. The procedure of the proposed SABS algorithm is illustrated in Algorithm 5.1.

In Algorithm 5.1, the algorithm starts by sorting all operations in a list according to their corresponding dependency levels. If two operations are in the same dependency level, operations with the nearest deadline first are prioritized. Next, a starting solution is initialized and is considered to be the current and the best solutions so far. Following that, the algorithm finds a new neighbored solution after which the objective value of the current and the new solutions are calculated and are compared. Accordingly, if the

acceptance probability of the new solution is greater than a random probability from 0 to 1, the new solution is accepted by the algorithm. Formally, the *acceptance probability* of a new solution S_{new} given a current solution S is defined in equation (36) [41]:

$$P(S, S_{new}, T) = e^{\left(\frac{E(S) - E(S_{new})}{T}\right)} \quad (36)$$

In equation (36), $E(S)$ represents the energy, objective value, of the solution S , and T is the temperature parameter of the simulated-annealing algorithm. Moreover, the algorithm keeps track of the best solution. In every iteration, the temperature is reduced based on the defined cooling rate. Eventually, the SABS algorithm stops when one of the following terminating criteria is satisfied:

- If there are no more neighbored solutions that can be acquired.
- If the temperature is cooled down completely, i.e. $temp \leq 1$.

As a result, the tracked best solution is considered as the final accepted solution. Before explaining the details of each step, the solution representation shall be defined in the next subsection.

```

Input: Temperature (temp), Cooling Rate (rate), List of Operations (operations), Vector of resources (resources)
Put operations data into a sort list ascendingly by dependency level
Set a starting solution (startSolution) = InitializeStartingSolution(operations, resources)
Set the current solution (currentSolution) as startSolution
Set the best solution (bestSolution) as currentSolution
Repeat
    currentSolution.Energy = GetObjectiveValue (currentSolution)
    newSolution = findNextSolution (currentSolution)
    newSolution.Energy = GetObjectiveValue (newSolution)
    // random number generation is uniformly distributed
    if acceptanceProbability (currentSolution.Energy, newSolution.Energy, temp) > rand_num()
        currentSolution = newSolution
    endif
    if currentSolution.Energy < bestSolution.Energy
        bestSolution = currentSolution
    endif
    temp = temp * (1 - rate)
until (a termination criterion is satisfied)
Solution = bestSolution

```

Algorithm 5.1: Simulated annealing-based scheduling algorithm

5.2.1. SABS solution representation. In this subsection, the encoding of the solution is presented. First, a resource block is defined as a compute block of a small cell in a specific time slot, formally (i, p, t) . Therefore, a solution is encoded as two vectors, a vector of resource blocks and a vector of operation finishing times. Each resource block stores information about its compute block, small cell, and time slot as well as the operation to which it is allocated. Furthermore, the vector of operation finishing times stores the time slots at which each operation is completed. The vector of resource blocks is represented in Figure 5.2.

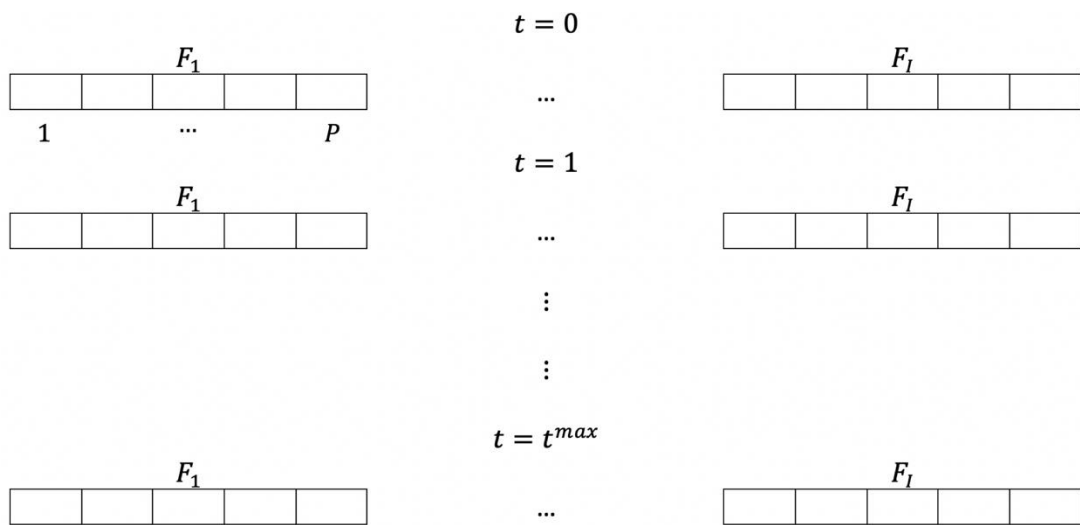


Figure 5.2: SABS solution representation

The representation illustrated in Figure 5.2 is proposed for any individual solution in order to respect all the scheduling problem constraints with the minimal number of checks in every iteration of SABS. In each resource block, an allocation can be denoted as j, k where $0 \leq j \leq J$ is the request number and $0 \leq k \leq K_j$ is the operation number of the request j . The representation of an allocation example is shown in Figure 5.3:

F_1				
1,1	1,1	2,3	2,3	2,3
1	2	3	4	5

Figure 5.3: Allocation representation (Example)

In the example illustrated in Figure 5.3, assuming $t = 0$, operation $k = 1$ of request $j = 1$ is allocated to the compute block $p = 1$ and $p = 2$ of the small cell $i = 1$ in the time slot $t = 0$, i.e. resource blocks $(i = 1, p = 1 \rightarrow 2, t = 0)$. Moreover, operation $k = 3$ of request $j = 2$ is allocated to the compute block $p = 3, p = 4$ and $p = 5$ of the small cell $i = 1$, i.e. $(i = 1, p = 3 \rightarrow 5, t = 0)$.

5.2.2. Initializing a starting solution. In the proposed SABS algorithm, a preliminary starting solution is initialized to start the search process by which better neighbored solutions are obtained. Moreover, to have a good starting solution, all the operations are sorted by their dependency level in advance. The process of obtaining a starting solution only checks for the dependency constraint. However, the starting solution is not necessarily a valid, feasible, solution as the deadline constraint can be violated in this step. This is acceptable since the search process of the SABS algorithm only explores feasible solutions. The process for initializing a starting solution is represented in Algorithm 5.2.

<p>Input: List of operations (list), Vector of resource blocks (resource) Output: A starting solution (solution) Create a new solution (solution) Set operation to be allocated (op) as the first operation in list <i>for</i> i <i>in</i> resources <i>do</i> <i>if</i> solution.allocation[i] <i>is free do</i> <i>if</i> isFit (op, i, solution) <i>and</i> DependencySatisfied (op, i, solution) <i>do</i> <i>for</i> $h = 0$ <i>to</i> op.requiredBlocks <i>do</i> solution.allocation[i] = op $i = i + 1$ // next resource block <i>endfor</i> <i>endif</i> solution.timeFinished[op] = i.timeslot op = op + 1 // point to the next operation in the list <i>endif</i> <i>endfor</i></p>

Algorithm 5.2: Algorithm for initializing a starting solution

Algorithm 5.2 schedules all operations preliminarily to start the search process of the SABS algorithm. By sorting the operations according to the corresponding dependency levels and deadlines, the algorithm tends to allocate operations parallelly

while prioritizing the nearest deadline first, with the minimum amount of calculations and constraint checks.

5.2.3. Getting the objective value. During every iteration in SABS, the objective values, energies, of a current solution and a new solution are calculated. Based on that, the acceptance function decides whether to accept the new solution or not. The energy, the objective value, of a solution is the summation of all requests' makespan. Algorithm 5.3 shows the procedure of the objective value calculation.

```

Input: Solution (solution)
Output: Objective Value (objVal)
objVal = 0, max = -1
for j in solution.requests do
    for op in j.operations do
        if max < solution.timeFinished[op] do
            max = solution.timeFinished[op]
        endif
    endfor
    objVal = objVal + max + 1
    max = -1
endfor

```

Algorithm 5.3: Algorithm for getting the objective value

5.2.4. Finding the next solution. After initializing a preliminarily starting solution, the search process starts by which new neighbored solutions are obtained. The procedure of getting a new neighbored solution is the state-of-the-art idea of the proposed SABS algorithm. First, the *degree of sequence* of an operation is defined as the number of consecutive time slots needed to execute the operation. For example, if an operation is fully parallelized such that it is executed in one time slot, the corresponding degree of sequence is defined to be “1”. Moreover, an operation’s degree of sequence is “2” if the operation is executed by half the number of the required compute blocks during two time slots. Accordingly, different possible solutions consist of operations that are allocated with various degrees of sequence. Algorithm 5.2 allocates all operations fully parallelized with a degree of sequence of “1”. Therefore, to find the next neighbored solution, the degree of sequence is incremented for one operation at a time. The amount of incrementation is dynamically decided such that operations can be shifted backward based on resource block availabilities to reduce the

makespan. The steps of finding a new neighbored solution are provided in algorithm 5.4. Algorithm 5.4 searches for the first free resource blocks, gap, in a given schedule. Afterward, the degree of sequence of the next allocated operation is incremented such that it fits into the gap and shifted back accordingly. The dynamic incrementation of an operation's degree of sequence, and the process of shifting the operation are performed by the function "sequence", to be discussed in detail in subsection 5.2.5. After the shifting process, the remaining operations that satisfy the dependency constraint are allocated in order, according to the sorted list of operations. Consequently, the algorithm ensures the allocation of all operations with the required number of resource blocks. Furthermore, the function denoted as "isFit", in Algorithm 5.4, ensures satisfying the constraint of executing each operation by only one small cell and the constraint of operations' non-preemption. The algorithm for this function shall be discussed later in subsection 5.2.6.

```

Input: Solution (solution), List of operations (list), Vector of resource blocks (resources)
Output: New Solution (newSolution)
Set operation to be allocated (op) as the first operation in list
SequentialSuccess = false // variable to check if sequencing was successful
newSolution = solution
for i in resources do
    if Solution.allocation[i] is free do
        if not SequentialSuccess do
            SequentialSuccess = sequence (newSolution, resources, i)
        else
            while op is not the last operation or not allocated and
            not isFit (op, i, newSolution) or
            not DependencySatisfied (op, i, newSolution) do
                op = op + 1 // next operation
            endwhile
        endif
        if op is not the last operation do
            for h = 0 to op.requiredBlocks do
                newSolution.allocation[i] = op
                i = i + 1 // next resource block
            endfor
            newSolution.timeFinished[op] = i.timeslot
            Op is set to the first unallocated operation
        endif
    endif
endfor

```

Algorithm 5.4: Finding the next solution

5.2.5. The sequence function. To find the next solution, operations need to be shifted backward by increasing the operations degree of sequence. This is handled by the algorithm represented as the function “sequence” in Algorithm 5.4. The procedure of the function “sequence” is presented in Algorithm 5.5. To shift an operation, the degree of sequence is increased dynamically. The incrementation of an operation’s degree of sequence depends on the number of required resource blocks of the operation and the number of available compute blocks to which the operation will be shifted. For example, if the number of consecutive available compute blocks, that the operation will be shifted to, is half the number of the required resource blocks, then the operation will be executed in two time slots. Consequentially, the degree of sequence is incremented to be “2”. Throughout the process of Algorithm 5.5, only the dependency and the deadline constraints are checked.

```

Input: Solution (solution), Vector of resource blocks (resources), A current resource (current_i)
Output: New Solution (newSolution), if the function is successful (success)
success = false // variable to check if the function is successfully executed
for i = current_i in resources do
    op = i.op
    if solution.allocation[i] is not free and op is not shifted do
        parallelBlocks = i.number_of_free_CBs
        sequentialBlocks = ceil (i.op.requiredBlocks /parallelBlocks)
        if (isFit (op, i, solution) and DependancySatisfied (op, i, solution) do
            clear all other allocations starting from i
            for d = 0 to sequentialBlocks do
                allocate the free “parallelBlocks” CBs from i with op
                i = i + (i.numOfCBs * i.numOfFogs) // next time slot
            endfor
            newSolution.timeFinished[op] = i.timeslot + sequentialBlocks - 1
            success = true
        endif
    endif
endfor
if not deadlineViolated(solution) do
    newSolution = solution
else
    success = false
endif

```

Algorithm 5.5: The sequence function

5.2.6. The “isFit” function. The function “isFit” is developed to indicate whether a set of available resource blocks is enough to allocate an operation with a

number of required blocks. The function “isFit” is presented in Algorithm 5.6. The Algorithm inputs are the operation information and a set of resource blocks.

```

Input: Solution (solution), Resource block (i), Operation (op)
Output: if the operation can fit into the given free blocks (fit)
Count = 0 // counter for free compute blocks
fit = false // variable to check if the operation can fit into the given free blocks
currentFog = i.fog
currentTime = i.timeslot
while currentTime = i.timeslot and currentFog = i.fog and solution.allocation[i] is free do
    count = count + 1
    i = i + 1 // next resource block
endWhile
if (count >= RequiredBlocks)
    fit = true
endif

```

Algorithm 5.6: The “isFit” function

5.2.7. Checking the dependency constraint. To avoid the violation of the dependency constraint, checking dependencies among operations of a request is a must during the shifting process. This task is handled by the developed “DependencySatisfied” function. The process of checking the dependency is shown in Algorithm 5.7. The dependency constraint for an operation k can be checked by comparing the finishing time of k with the finishing time of each operation h that k depends on, i.e. $\gamma_{jkh} = 1$, where $0 < k, h \leq K_j$. Accordingly, the dependency constraint is satisfied only if operation h is completed before the execution of operation k starts.

```

Input: Solution (solution), Resource block (i) Operation to be checked (op)
Output: if the dependency is satisfied (isDependencySatisfied)
isDependencySatisfied = true // variable to check if the dependency is satisfied
for o in op.dependencies do
    if (solution.timeFinished[o] >= i.timeslot or o is not allocated)
        isDependencySatisfied = false
    endif
endfor

```

Algorithm 5.7: Dependency constraint check

5.2.8. SABS algorithm: complexity analysis. Algorithm 5.1 involves initializing a starting solution, given in Algorithm 5.2. To acquire such solution, Algorithm 5.2 allocates all operations and checks the dependency constraint for each operation, correspondingly giving a time complexity of $O(n^2)$, where n is the number of operations. Following that, in Algorithm 5.1, the search for the next solution begins. The simulated annealing algorithm iterates $O(\log n)$ temperature steps. In each iteration the next solution is obtained as presented in Algorithm 5.4. The algorithm for finding the next solution iterates for every resource block ($O(n)$). For each available resource block, the dependency constraint is checked for the neighbored allocated operation ($O(n)$). Therefore, the complexity of Algorithm 5.4 is $O(n^2)$. Proceeding with Algorithm 5.1, if the new solution is rejected, the algorithm starts looking for another solution. Therefore, the new solution is rejected in constant time, i.e. $O(1)$. On the other hand, the time complexity of updating the current and the best solution after accepting a new solution is $O(n)$. Consequently, the time complexity for the proposed SABS algorithm, presented in Algorithm 5.1, is $O(n^2 + n^3 \log n) = O(n^3 \log n)$.

5.2.9. SABS algorithm: example. In this subsection, the example previously used in Chapter 3 to validate the scheduling model represented in equations (9-15) is solved using the proposed SABS heuristic algorithm for further understanding. Moreover, the list of operations in the considered example is presented in Table 5.1:

Table 5.1: All operations before sorting

$[j, k]$	IPS * 10^5	τ	Dependency level
[1,1]	2	3	0
[1,2]	4	3	1
[1,3]	5	3	2
[2,1]	3	3	0
[2,2]	2	3	1
[2,3]	4	3	2
[3,1]	4	3	0
[3,2]	4	3	1
[3,3]	3	3	2

First, all operations are grouped by their corresponding dependency levels. Afterward, the operations in each group are sorted by the corresponding deadlines. This allows fast preliminary allocation of all operations while avoiding complex

computations and constraint violations. Eventually, all the operations are sorted in the order depicted in Table 5.2.

Table 5.2: All operations after sorting

$[j, k]$	IPS * 10^5	τ	Dependency level
[1,1]	2	3	0
[2,1]	3	3	0
[3,1]	4	3	0
[1,2]	4	3	1
[2,2]	2	3	1
[3,2]	4	3	1
[1,3]	5	3	2
[2,3]	4	3	2
[3,3]	3	3	2

According to Algorithm 5.2, a solution is initialized by considering the maximum parallelism while allocating all the operations. This means that the algorithm maximizes the number of compute blocks parallelly allocated for each request at any given time such that it executes during the minimum number of time slots. Following the order established in Table 5.2, the allocation during the time slot $t = 0$ is shown in Figure 5.4.

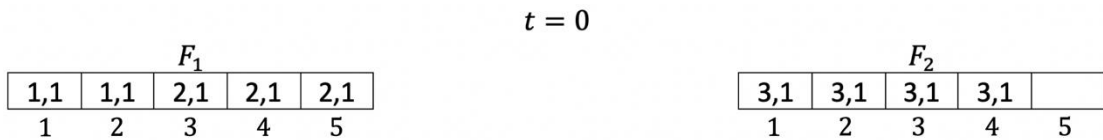


Figure 5.4: Allocation for time slot $t = 0$ (starting solution)

In Figure 5.4 all operations fit into the available compute blocks such that it executes within one time slot according to Algorithm 5.6. After allocating each of $O_{1,1}$, $O_{2,1}$ and $O_{2,1}$, the information regarding their finishing times is updated. Since operation $O_{1,2}$ requires four compute blocks to be executed in one time slot and there is only one compute block remaining during the current time slot $t = 0$, the allocation of the operation $O_{1,2}$ shall be shifted to the next time slot $t = 1$. Shown in Figure 5.5.

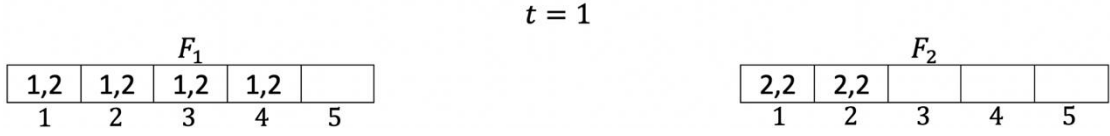


Figure 5.5: Allocation for time slot $t = 1$ (starting solution)

Since the dependency levels of $O_{1,2}$ and $O_{2,2}$ are not zero, i.e. they depend on other operations, the finishing time of each operation that $O_{1,2}$ and $O_{2,2}$ depend on must be checked according to Algorithm 5.7. Operations $O_{1,2}$ and $O_{2,2}$ depend on operations $O_{1,1}$ and $O_{2,1}$, respectively. Since the execution of $O_{1,1}$ and $O_{2,1}$ is finished at time slot $t = 0$, operations $O_{1,2}$ and $O_{2,2}$ are allocated at time slot $t = 1$ as shown in Figure 5.5. Afterward, $O_{3,2}$ is considered for allocation according to the ordered list. Since $O_{3,2}$ requires four resource blocks and there are only three available consecutive resource blocks, the allocation of $O_{3,2}$ is shifted to the next time slot $t = 2$. Shown in Figure 5.6.

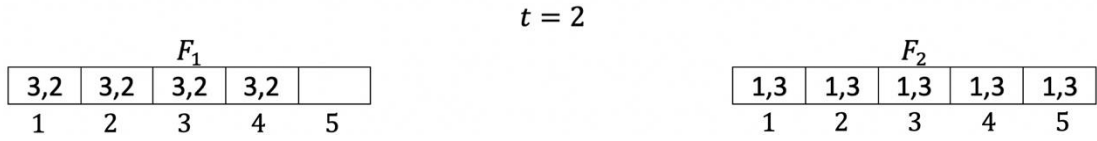


Figure 5.6: Allocation for time slot $t = 2$ (starting solution)

According to Algorithm 5.6, $O_{3,2}$ can fit in the available compute blocks of the small cell $i = 1$ at time slot $t = 2$. Furthermore, the dependency constraint is checked by examining the finishing time of each operation that $O_{3,2}$ depends on, i.e. operation $O_{1,2}$. Since operation $O_{1,2}$ is completed at the time slot $t = 1$, which is before the currently considered time slot $t = 2$, then the dependency constraint is satisfied according to Algorithm 5.7. As a result, $O_{3,2}$ is allocated and the information of its finishing time is updated accordingly. Likewise, $O_{1,3}$ is allocated at the same time slot $t = 2$. Subsequently, $O_{2,3}$ is considered for the next allocation, according to the ordered list. Since $O_{2,3}$ requires four resource blocks and there are no available compute blocks during the current time slot $t = 2$, the allocation of $O_{2,3}$ is shifted to the next time slot $t = 3$ shown in Figure 5.7.

$t = 3$									
F_1					F_2				
2,3	2,3	2,3	2,3		3,3	3,3	3,3		
1	2	3	4	5	1	2	3	4	5

Figure 5.7: Allocation for time slot $t = 3$ (starting solution)

Similarly, the dependency constraint is checked by examining the finishing time of each operation that $O_{2,3}$ depends on, i.e. operation $O_{2,2}$. Since operation $O_{2,2}$ finished at the time slot $t = 1$, that is before the current time slot $t = 3$, then the dependency constraint is satisfied. Consequently, $O_{2,3}$ is allocated and the information of its finishing time is updated accordingly. Finally, the operation $O_{3,3}$ is allocated in a similar fashion. As a result, an initial allocation is obtained by which the proposed SABS algorithm starts the search process, illustrated in Figure 5.8.

$t = 0$									
F_1					F_2				
1,1	1,1	2,1	2,1	2,1	3,1	3,1	3,1	3,1	
1	2	3	4	5	1	2	3	4	5
$t = 1$									
F_1					F_2				
1,2	1,2	1,2	1,2		2,2	2,2			
1	2	3	4	5	1	2	3	4	5
$t = 2$									
F_1					F_2				
3,2	3,2	3,2	3,2		1,3	1,3	1,3	1,3	1,3
1	2	3	4	5	1	2	3	4	5
$t = 3$									
F_1					F_2				
2,3	2,3	2,3	2,3		3,3	3,3	3,3		
1	2	3	4	5	1	2	3	4	5

Figure 5.8: The initial allocation for sabs algorithm

Following Algorithm 5.1, the objective value of the initial solution is calculated to be 11 time slots. This value is calculated by Algorithm 5.3. Afterward, Algorithms 5.4 and 5.5 are used to find the next neighbored solution. Accordingly, operation $O_{1,2}$ cannot be shifted since the deadline constraint would be violated. On the other hand, operation $O_{2,2}$ is shifted by increasing its degree of sequence to “2” because two compute blocks are required to execute the operation and there is one available compute block ahead of its allocation. As a result, $O_{2,2}$ is allocated to the available resource

blocks ($i = 1, p = 5, t = 1 \rightarrow 2$). Subsequently, the allocation of the remaining operations proceeds according to the ordered list of operations. This is similar to the initialization of the starting allocation. The dependency constraint for operation $O_{3,2}$ is checked then the operation is allocated to the resource blocks ($i = 2, p = 1 \rightarrow 4, t = 1$). Operation $O_{1,3}$ does not fit in the resource blocks ($i = 2, p = 5, t = 1$) or ($i = 1, p = 1 \rightarrow 4, t = 2$). However, $O_{1,3}$ fits in the available blocks ($i = 2, p = 1 \rightarrow 5, t = 2$) in which it is allocated accordingly. The first feasible allocation for $O_{2,3}$ and $O_{3,3}$ are ($i = 1, p = 1 \rightarrow 4, t = 3$) and ($i = 1, p = 1 \rightarrow 3, t = 2$), respectively. The resultant allocation is presented in Figure 5.9.

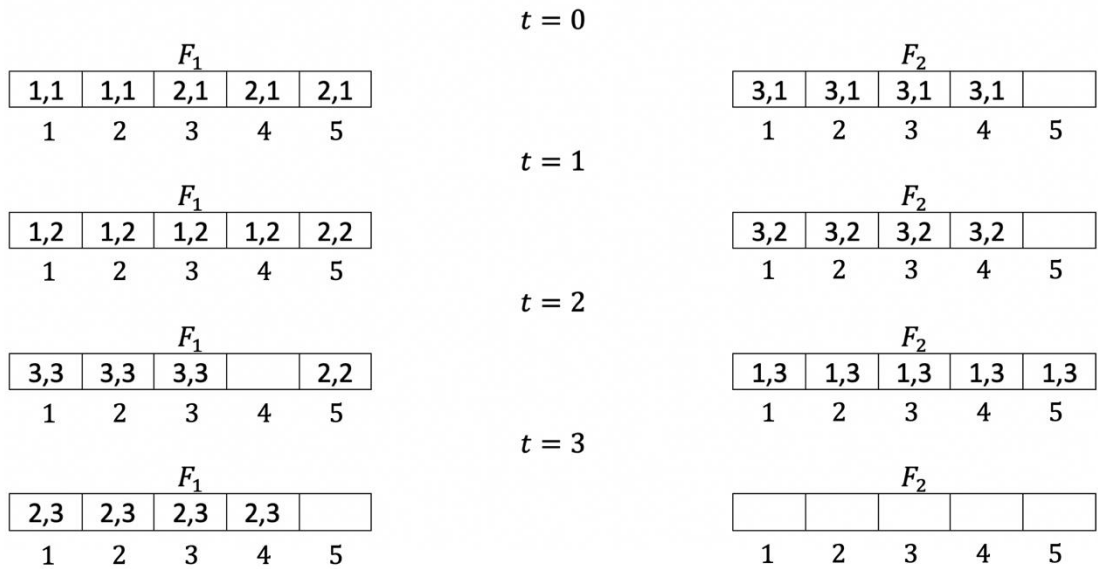


Figure 5.9: The allocation after finding the next solution

By applying Algorithm 5.3, the objective value of this solution is found to be 10 time slots. The algorithm continues the search to get better solutions by checking each available resource block and shift the neighbored operation if possible. Accordingly, the next available resource block to be checked is ($i = 2, p = 5, t = 1$). However, operation $O_{3,3}$ cannot be shifted due to the dependency relationship it has with $O_{3,2}$. Afterward, the algorithm proceeds to the next available resource block ($i = 1, p = 4, t = 2$). Since $O_{2,2}$ is already shifted, according to Algorithm 5.5, the search continues by checking the next available resource block ($i = 1, p = 5, t = 3$). By considering this block, there are no more operations to be shifted, therefore no more

solutions can be obtained. This is one of the terminating criteria. Consequently, the search process is terminated, and the best objective value is found to be “10”. This conforms with the objective value acquired by solving the optimization model (9-15) using CPLEX optimization engine. Therefore, this confirms that the proposed SABS algorithm solved the considered simple scheduling problem instance with the minimum objective value.

5.3 Auction-Based Winner Determination (ABWD) Algorithm

According to the proposed decentralized framework presented in Section 4.2, after the self-classification and local scheduling processes, provider agents will be sharing available resources and consumer agents will be handing over requests to be executed by provider agents. In Chapter 4, considering the decentralized nature of the problem, the decision-making involved in the coordinated control is represented as the winner determination problem. Furthermore, the winner determination problem is modeled as a combinatorial auction problem that involves calculations of agent preferences represented as utility values. However, due to the tremendously large possible combinations of available resources, preferences calculation and winner determination are NP-Hard problems. Thus, a novel developed ABWD algorithm is proposed to obtain an adequate solution in polynomial time.

5.3.1. Preferences approximation. In Chapter 4, according to the proposed framework, consumer agents should calculate their preferences that correspond to all possible combinations of available resources. In addition, provider agents need to count all their available resource blocks and the corresponding preferences. Such complex computations affect performance dramatically. Therefore, in the proposed ABWD algorithm, consumer and provider agents approximate their preferences. Accordingly, consumers encode their preferences, bids, for each request as its requirements, i.e. number of operations, instruction count and dependencies. Such information can be retrieved from requests’ metadata without extensive computations. Moreover, to uniquely identify the bid for each handed-over request, each bid is referred to by the ID of its corresponding request and the ID of its corresponding consumer agent. For example, agent $i = 2$, shown in Figure 4.6, classifies the request $j = 3$ as a handed-over request. Therefore, the bid corresponding to the request $j = 3$ shall be referred to as $\{2,3\}$. On the other hand, provider agents encode their available resources as the

number of compute blocks they have, and the time slot from which all the compute blocks are available. For further understanding, consider the example agent depicted in Figure 4.8. The encoded data regarding the shareable available resources of that agent is illustrated in Figure 5.10.

In Figure 5.10, the area shaded in red represents the encoded information of a provider's available resources to be sent to the coordinator. Such information is encapsulated in three values: the corresponding provider agent's ID, the time slot from which all the compute blocks are available, and the provider agent's capability in terms of number of compute blocks. For example, in Figure 5.10, all the resources of the agent $i = 3$ are available starting from the time slot $t = 3$, and it has the capability of six compute blocks. Consequently, the encapsulated information of the shareable available resources can be represented as $\{3,3,6\}$. Afterward, the encoded preferences for both providers and consumers are shared with the coordinator by which the auction takes place for winner determination.

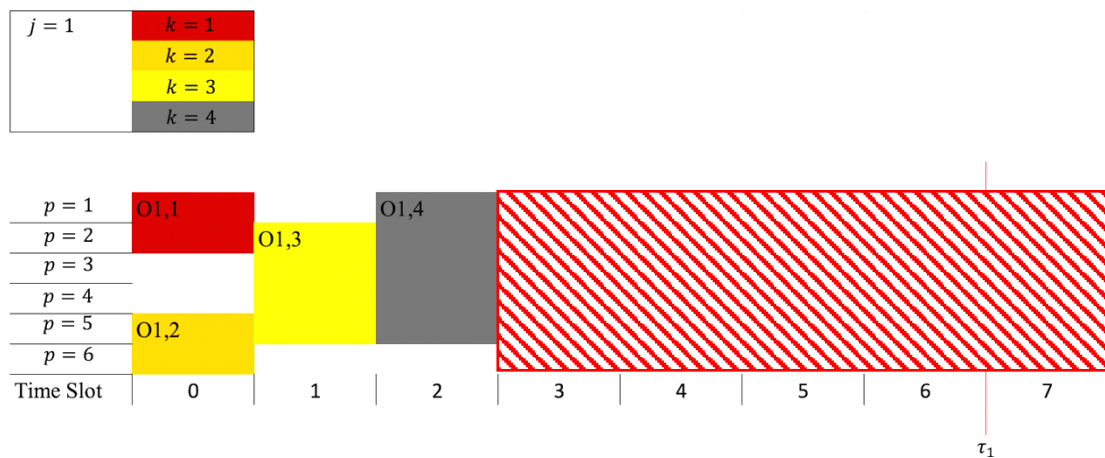


Figure 5.10: The available resources encoded by a provider agent (shaded area)

5.3.2. Winner determination and solution representation. After receiving the encoded preferences from provider and consumer agents by the coordinator, two lists are created accordingly: A list of goods that holds all the shareable available resources, and a list of bids that represents handed-over requests' preferences. After constructing the two lists, the auction starts in order to determine the winners. As a result, the outcome would be a list of winner-determinations. Particularly, each winner-

determination is represented as a pair of a winner handed-over request's bid and its corresponding earned good, i.e. resources. The inputs and outputs of the winner determination process are represented in Figure 5.11.

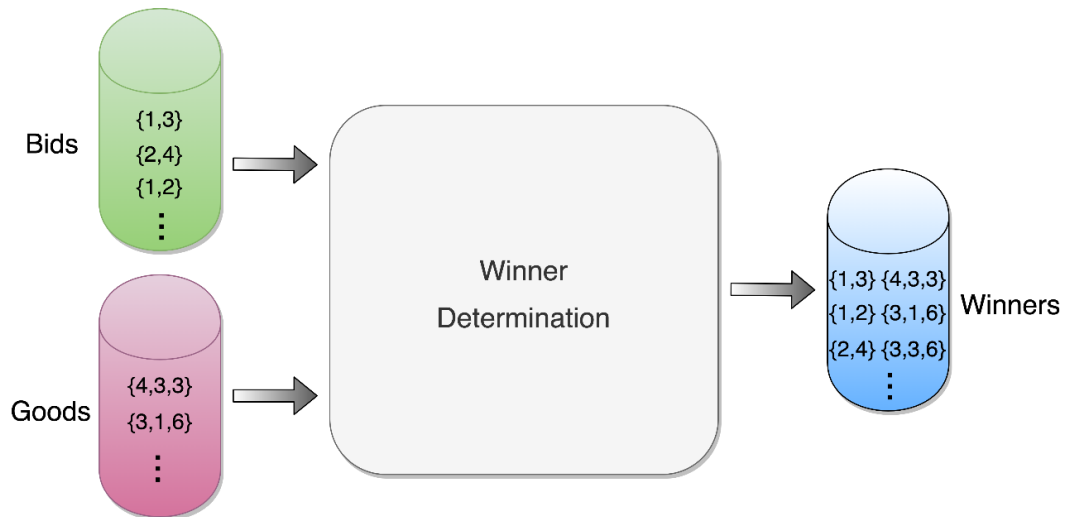


Figure 5.11: Inputs/outputs of the winner determination algorithm

Figure 5.11 shows the inputs for the winner determination process, a list of goods and a list of bids. Additionally, it shows the output of the process which is a list of winner-determinations, winning bids and the corresponding earned goods. Furthermore, in Figure 5.11, the resources of agent $i = 3$, namely $\{3,1,6\}$, are sold to two different bids. Specifically, the bid $\{1,2\}$ wins the goods that represent the resource blocks of agent $i = 3$ starting from time slot $t = 1$ while the bid $\{2,4\}$ wins goods that represent the resources of the same agent, $i = 3$, starting from time slot $t = 3$. After the winner determination process, the coordinator sends the winner requests to the corresponding providers according to the list of winner-determinations.

As mentioned earlier in subsection 5.3.1, all providers send approximated information regarding their available resources, i.e. only resource blocks during the time slots in which all compute blocks are available. Thus, this degrades the auction results' accuracy. To compensate, a dynamic sealed-bid first-price auction mechanism is imposed, observable in Figure 5.1.

5.3.3. Winner determination algorithm. In this subsection, the algorithm for solving the winner determination problem is presented. In auctions, consumer agents

are the buyers and provider agents are the sellers. The sealed-bid first-price auction starts by offering one good at a time to all the buyers. For each good, all buyer bids and the seller's encoded preferences are used to calculate the utility value. Subsequently, the bid that results in the highest utility value wins the offered good. Afterward, the winning bid is removed from the auction and the remaining resources of the seller are estimated. If all the resources of the offered good are estimated to be allocated, then the offered good is removed from the auction and the next good, according to the list of goods, is offered in the market. Finally, the auction repeats until all the submitted bids have won some goods, or all goods have been sold. The process of winner determination is depicted in Algorithm 5.8.

```

Input: List of Goods (goods), List of Bids (bids)
Output: List of Winners (winners), Variable indicates if all bids won some good (done)
done = false
currentGood = firstGood (goods)
While goods and bids are not empty do
    maxUtility = -inf
    for i in bids do
        currentUtility = CalculateUtility (i, currentGood)
        if currentUtility > maxUtility do
            maxUtility = currentUtility
            theWinnerBid = i
        endif
    endfor
    theWinnerGood = currentGood
    winnerPair = {theWinnerBid, theWinnerGood}
    adds the "winnerPair" to the list "winners"
    erase "theWinnerBid" from the list "bids"
    update what remains from the "currentGood"
    if currentGood has no more resources do
        erase "currentGood" from the list "goods"
    endif
    currentGood = next (goods)
endwhile
if bids is empty do
    done = true
endif

```

Algorithm 5.8: Winner determination

There is a drawback in the proposed Algorithm 5.8. To explain the issue, consider the following example: let there be two handed-over requests, i.e. $j = 1$ and $j = 2$, and two provider agents, namely $i = 1$ and $i = 2$. Also, assume that request $j =$

1 has greater resource requirements than request $j = 2$. Furthermore, let agent $i = 1$ have more shareable available resources than agent $i = 2$ such that agent $i = 1$ can accommodate only one of the two requests while agent $i = 2$ can accommodate only the smaller request, $j = 2$. Moreover, assume that the smaller request $j = 2$ has a higher willingness-to-pay. Hence, if agent $i = 1$ resources are firstly offered in the auction, the smaller request $j = 2$ will be executed by agent $i = 1$. However, agent $i = 1$ is capable of executing only one of the two requests and agent $i = 2$ is only capable of executing $j = 2$. Consequentially, both agents will not be able to execute the larger request $j = 1$ and the scheduling process will fail. To tackle this issue, the list of goods is sorted in descending order according to their resource availabilities before being offered in the auction. Accordingly, goods with a limited number of shareable available resource blocks are offered first to be purchased by the requests that require fewer resources.

5.3.4. Utility calculation. In this subsection, the algorithm for calculating the utility is presented. The algorithm is developed such that the utility value is influenced by the deadline and the estimated finishing time of the buyer's request, and the seller's estimated shareable available resources. Accordingly, the earlier a request is estimated to finish its execution, the more its buyer is willing to pay. This reflects buyer preferences. Furthermore, the more resources required for a request, the more its buyer is willing to pay. This reflects seller preferences. The algorithm for calculating the utility is defined as the "CalculateUtility" function in Algorithm 5.8. The function procedure is presented in Algorithm 5.9.

<p>Input: Good (good), Bid (bid) Output: Utility Value (utility) utility = -inf // no deal estimatedExecutionTime = max(ceil(bid.requiredBlocks/good.numOfCBs), bid.dependencyLevel) estimatedFinishingTime = estimatedExecutionTime + good.startTimeSlot - 1 neededResources = bid.requiredBlocks/good.numOfCBs if bid.deadline > estimatedFinishingTime do // if deadline is met, check utility // calculate the utility utility = neededResources + 2*timeWindow - estimatedFinishingTime - bid.deadline endif</p>

Algorithm 5.9: Utility calculation

In Algorithm 5.9, the estimated finishing time of a request is the time in which the execution starts plus the estimated execution time of that request. To estimate the execution time of a request, the number of time slots needed to execute the request and the maximum dependency level of the request are compared. Accordingly, the execution time of the request is estimated to be the maximum of the two compared values. Furthermore, if the estimated finishing time of a request passes its deadline, then the auctioneer rejects the bid and the utility value is considered to be $-inf$. The simplicity of the estimation reduces the complexity dramatically, however, it leads to degradation in solution accuracy. To counterweigh, the auction is performed dynamically as shown in the workflow of the proposed solution presented in Figure 5.1. In detail, after provider agents receive handed-over requests according to the winners list, they locally schedule the handed-over requests. Afterward, the information regarding providers' shareable available resources is updated accordingly and sent to the auctioneer. Correspondingly, an auction is conducted offering the updated list of goods. The whole process is repeated until all handed-over requests are allocated to some shareable available resources.

5.3.5. ABWD algorithm: example. In this subsection, an example is presented and is solved using the proposed ABWD algorithm for further understanding. First, lists of resources and handed-over requests are defined. Accordingly, the considered example consists of three agents: two of type providers and one of type consumer. The two providers have the same compute capabilities of $P = 6$ but a different number of shareable available resources. The list of resources, sorted in descending order according to their availabilities, is represented in Table 5.3. Furthermore, the consumer agent has two requests to hand over. The list of handed-over requests is presented in Table 5.4.

After setting up both lists, the coordinator shall start the auction. According to the sorted list of goods, the first good to be offered is $\{3,5,6\}$. Complying to Algorithm 5.9, the utility for selling the good $\{3,5,6\}$ considering the bid $\{2,5\}$ and $\{2,4\}$ can be calculated as "5" and "7", respectively. The calculation details are provided in Appendix C. The winning bid of the good $\{3,5,6\}$ is chosen to be $\{2,5\}$ since it results in the greatest utility value. The execution time of $\{2,5\}$ is estimated to be two time slots. Therefore, for the good $\{3,5,6\}$, the starting time from which all its compute

blocks are available is updated to be “8” instead of “5”. Accordingly, the good $\{3,5,6\}$ is updated to be $\{3,8,6\}$.

Table 5.3: List of resources

i	Free Starting Time	Number of CBs	Notation
3	5	6	$\{3,5,6\}$
1	3	6	$\{1,3,6\}$

Table 5.4: List of handed-over requests

$\{i, j\}$	k	IPS * 10^5	τ	Dependency level
$\{2,5\}$	1	2	6	0
	2	3		1
	3	4		2
$\{2,4\}$	1	4	8	0
	2	2		0
	3	4		1
	4	5		2

Following Algorithm 5.8, the bid $\{2,5\}$ is removed from the auction and the succeeding good, according to the ordered list of goods, is offered next. Similarly, the utility of selling $\{1,3,6\}$ considering the bid $\{2,4\}$ is calculated to be “5”. The calculation details are provided in Appendix C. The bid $\{2,4\}$ is providing the highest utility since it is the only remaining one in the auction, hence, it is the winner of the good $\{1,3,6\}$. According to Algorithm 5.8, $\{2,4\}$ is removed from the auction. Afterward, according to the list of winners, the coordinator sends the handed-over requests to their corresponding provider agents for local scheduling and execution. Subsequently, each agent schedules the received requests and inform the coordinator with the updated shareable available resources accordingly. Consequently, the auctioneer initiates another auction to schedule the rest of the handed-over requests, if any. In this example, there are no more handed-over requests to bid in the auction. Hence, the algorithm is terminated.

Chapter 6. Experimentation and Results

In this chapter, the proposed model is implemented using the CPLEX optimization tool to get an exact solution. Furthermore, the proposed heuristic technique is implemented in C++ to obtain an approximated solution. Afterward, a comparison is conducted between the exact solution and the approximated solution to validate the quality of the proposed heuristic technique. For SABS and ABWD algorithms, sensitivity analysis is performed to examine the impact of each system parameter on the overall solution. Despite the decentralized nature of the fog computing environment, a centralized scheduling approach, assuming full knowledge of the 5G network and agent capabilities, is considered for comparative purposes. The assumed centralized approach is used as a benchmark and is compared to the proposed decentralized approach in terms of solution quality and execution time.

6.1. Data Generation

The data used for experimentation is generated by a data generator developed and implemented in C++. The output format of the developed data generator is compatible with the heuristic algorithm and the CPLEX optimization tool. In addition, different parameters can be set such as the number of small cells, the number of compute blocks for each small cell, the capability of each compute block, the number of requests to be generated, and the minimum and maximum numbers of operations per request. Furthermore, multiple system parameters are randomized. To make the data generation process as realistic as possible, different distributions are used to randomly generate various parameters. Specifically, the instruction count per operation is generated randomly following a uniform distribution. However, since the number of operations per request is an unexpected random number between the user-specified min-max bounds, it is randomly generated following a Poisson distribution [4], [5].

6.2. Centralized Approach Experimentations

In this section, multiple experiments considering the centralized approach are presented. Different numbers of resources and operations are considered to assess system performance. Moreover, sensitivity analysis is performed to address the impact of each input parameter on the system behavior. All experimentations are conducted on a machine that consists of Intel(R) Xeon(R) Gold 5118 processor with a speed of 2.3

GHz and a RAM of 48 GB. The 64-bit version of Microsoft Windows 10 Education is the operating system installed.

6.2.1. CPLEX simulation results. The proposed optimization model for the centralized approach, represented in equations (9-15), is implemented in the optimization programming language (OPL) and solved using the constraint programming (CP) engine in the IBM ILOG CPLEX optimization studio. The parameters used in the simulation process are the same parameters used to validate the model, shown in Table 3.3. Using the developed data generator, different numbers of requests, operations, small cells and compute blocks are considered. Table 6.1 shows information regarding all the conducted experimentations using CPLEX to solve the optimization problem assuming centralized settings.

Table 6.1: CPLEX experiments for the centralized approach

Experiment No.	Number of Requests	Number of Operations	Number of Small Cells	Number of Compute Blocks
1	2	6	2	8
2	3	9	2	11
3	3	11	2	20
4	4	16	2	20
5	3	38	3	22

For each experiment in Table 6.1, an optimal solution is obtained using the CPLEX optimization tool. Furthermore, the requests' average makespan, in time slots, and the execution time of the model solver, in seconds, are logged and presented in Table 6.2. It is noteworthy that the execution time does not include the communication delay involved in solving the scheduling problem in all the upcoming experiments.

Table 6.2: CPLEX results for the centralized approach

Experiment No.	Average Makespan (time slots)	Execution Time (s)
1	2	1
2	2.33	5
3	3	80
4	3.5	8350
5	N/A	N/A

To interpret the scheduling problem complexity, Figure 6.1 presents a bar chart that shows the impact of each experiment on the execution time of the model solver.

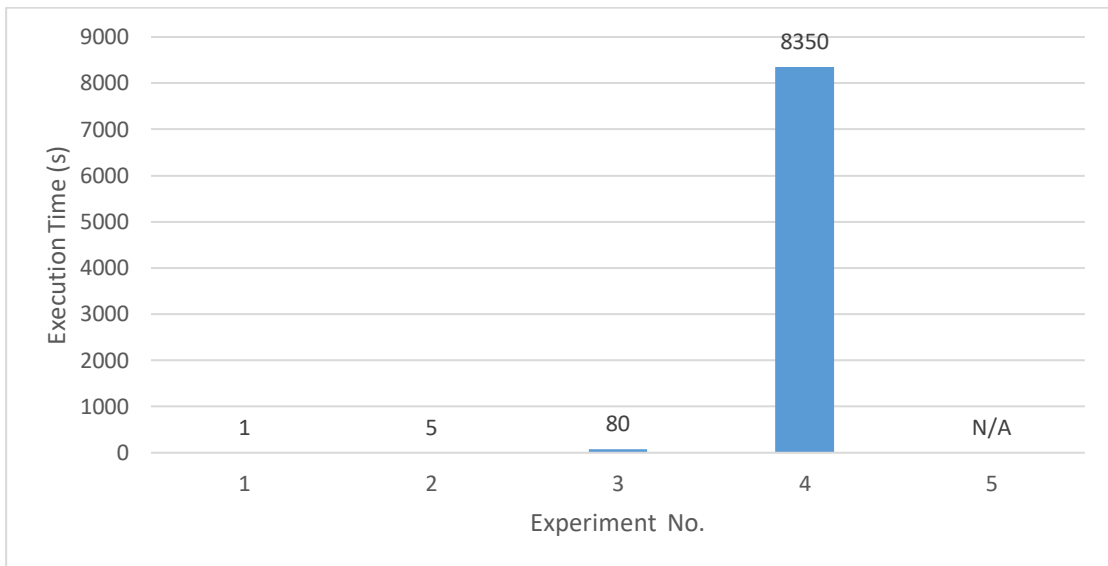


Figure 6.1: Centralized approach execution time (CPLEX)

From the bar chart in Figure 6.1, it is observable that by increasing the search space, whether by increasing the number of operations or the number of compute blocks, the execution time is increasing dramatically. This complies with the fact that the scheduling problem is an NP-Hard in the strong sense. Additionally, the influence on the average makespan can be perceived in the bar chart shown in Figure 6.2.

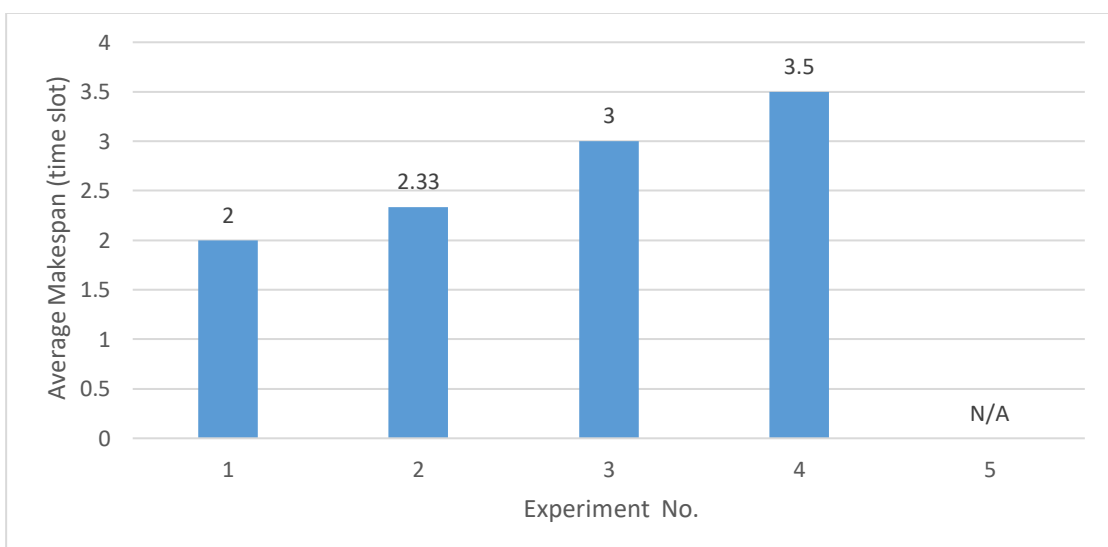


Figure 6.2: Centralized approach average makespan (CPLEX)

By considering experiment No. 3 and 4 in Figure 6.2, it is observable that the number of compute blocks is the same, 20 compute blocks. However, the number of operations increased from 11 to 16 and the average makespan increased from 3 to 3.5. Consequently, the impact of increasing the number of operations on the resultant average makespan becomes questionable. Therefore, sensitivity analysis is conducted and presented later in this section.

6.2.2. SABS algorithm performance evaluation. The proposed SABS algorithm is implemented in C++ to solve the scheduling problem. In this subsection, the proposed solution is tested, and the results are compared with the optimal solutions obtained by CPLEX. For comparative purposes, the same generated data, shown in Table 6.1 is used. Likewise, system parameters used for the model validation and for the CPLEX experimentation, presented in Table 3.3, are considered. The results of running the simulation of the SABS algorithm are depicted in Table 6.3.

Table 6.3: SABS results for the centralized approach

Experiment No.	Average Makespan (time slots)	Execution Time (ms)
1	2	0.128
2	2.33	0.285
3	3.67	0.360
4	3.5	0.760
5	8.67	1.224

In Table 6.3, the average makespan increases with the escalation of the problem size. To be able to compare the results with the optimal solutions, the average makespan obtained from the CPLEX simulation and SABS results are illustrated in Figure 6.3. The average makespan obtained by the proposed SABS algorithm is very close to the average makespan acquired from solving the model by CPLEX engine. However, some results are slightly greater than the optimal solution. Nonetheless, these results are acceptable considering the amount of time saved in solving the scheduling problem.

Thus far, all the experiments were conducted considering a small number of inputs, i.e. requests, operations, small cells and compute blocks, for the purpose of comparing the proposed SABS algorithm with the optimal solution. However, to evaluate the performance of the proposed algorithm in more realistic scenarios, more

experiments are performed considering problems with various set-ups and scaled sizes. The experimentation parameters, resultant average makespan and execution time are presented in Table 6.4.

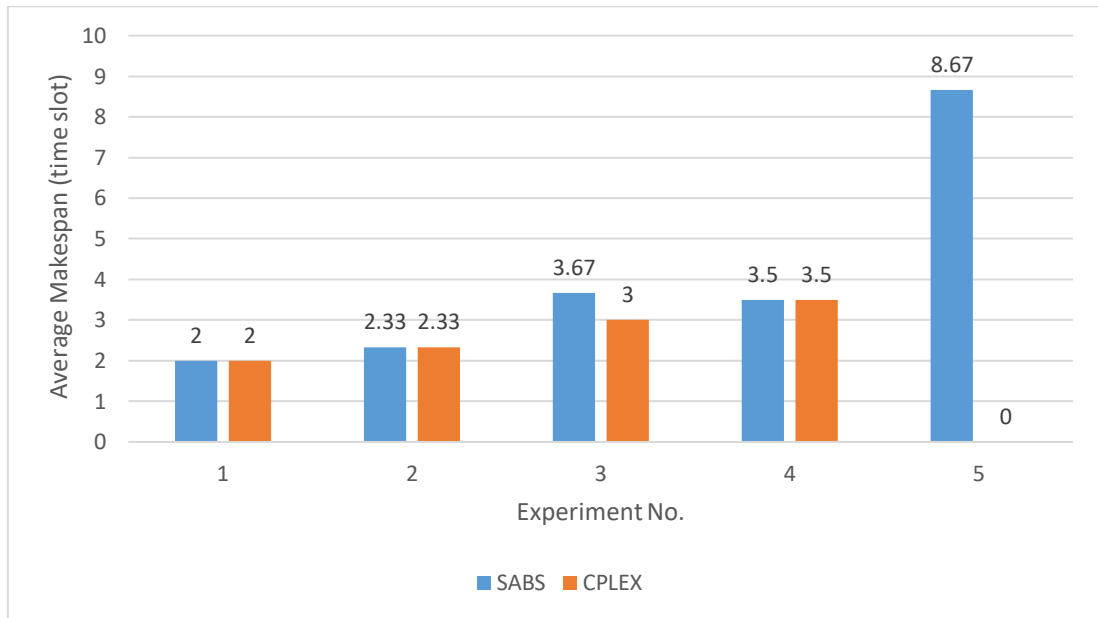


Figure 6.3: SABS vs. CPLEX (average makespan)

Table 6.4: SABS experimentation parameters and results

Experiment No.	Number of Requests	Number of Operations	Number of Small Cells	Number of Compute Blocks	Average Makespan (time slot)	Time Elapsed (ms)
1	2	6	2	8	2	0.128
2	3	9	2	11	2.33	0.285
3	3	11	2	20	3.67	0.36
4	4	16	2	20	3.5	0.76
5	4	16	3	23	4	0.814
6	3	38	3	22	8.67	1.224
7	100	500	5	100	22.01	3.201
8	20	1000	2	100	48.8	12.631
9	100	1000	10	200	22.17	8.569
10	100	2000	5	100	94.39	13.924
11	100	2000	20	400	24.14	16.158

In Table 6.4, the problem size is ascendingly increasing throughout the experiments. Moreover, the average makespan escalates by increasing the problem size.

To further examine the influence in the average makespan, a bar chart shows the average makespan corresponding to each experiment is shown in Figure 6.4.

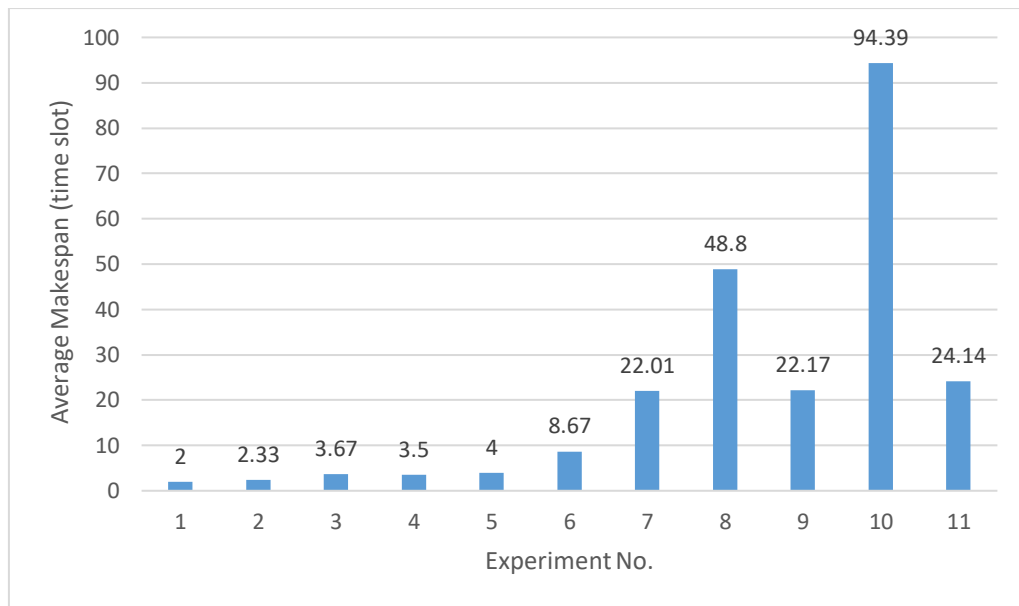


Figure 6.4: SABS average makespan

In the bar chart in Figure 6.4, it is observable that the greater the size of the problem, the larger the average makespan. Additionally, by examining experiments 7, 8 and 10, one can notice the impact of increasing the number of operations on the average makespan while keeping the other parameters fixed. In other words, whenever the number of operations increases, the average makespan increases. However, the average makespan drops in some experiments, namely 4, 9 and 11. In experiment 4, the number of requests increases. Therefore, the requests' average makespan decreases. On the other hand, the reason behind the drop noticeable in experiment 9 and 11 is different. Precisely, the number of compute blocks are incremented significantly in these two experiments. Consequently, there are more available compute blocks that enable more parallel executions of operations. In Figure 6.4, the effecting parameter might be the number of operations or the number of compute blocks or other factors. To confirm the influential parameter, sensitivity analysis is presented later in this section.

Similarly, by examining the change in execution time throughout the experiments in Table 6.4, one can notice that the execution time rises proportionally to

the problem size. For further interpretation, the change in execution time of each experiment is illustrated in the graph shown in Figure 6.5.

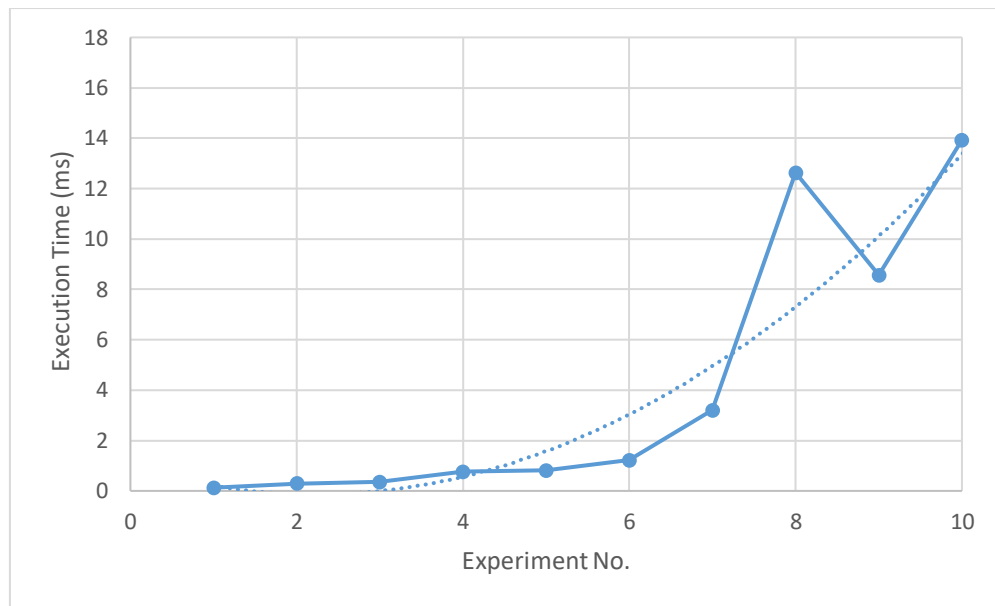


Figure 6.5: SABS execution time

In Figure 6.5, the execution time of the SABS algorithm escalates as the problem size increases. However, the execution time of experiment 8 does not follow the overall trend demonstrated by the other experiments. This might be a result of having a lower number of requests, or fewer number of small cells, or other hidden factors. In fact, the effect of different parameters is unclear considering the results obtained from the conducted experimentations. Therefore, this stimulated the need for carrying out sensitivity analysis in which each parameter's impact on system performance is investigated.

6.2.3. SABS sensitivity analysis. To examine the influence of a specific parameter on the average makespan and the execution time of the proposed SABS algorithm, all the other parameters are fixed while varying the value of that specific parameter. Accordingly, to find the effect of the number of requests in a given scheduling problem, experiments with various numbers of requests are conducted. For each experiment, the corresponding average makespan and execution time are logged. All experiments' parameters and results are presented in Table 6.5.

Table 6.5: The impact of varying the number of requests on SABS algorithm

Experiment No.	Number of Requests	Number of Operations	Number of Small Cells	Number of Compute Blocks	Average Makespan (time slot)	Execution Time (ms)
1	2	100	2	50	25	15.424
2	5	100	2	50	11.4	8.679
3	10	100	2	50	9.9	1.569
4	20	100	2	50	9	0.969
5	50	100	2	50	4.98	0.841
6	100	100	2	50	3.28	0.226

In Table 6.5 the average makespan decreases as the number of requests increases. Since the total number of operations is unchanging, the amount of instructions that need to be executed by the small cells is the same in all the experiments. However, the number of operations per request varies throughout the experiments. For further analysis, the relationship between the number of operations per request and the average makespan can be illustrated in the bar chart shown in Figure 6.6.

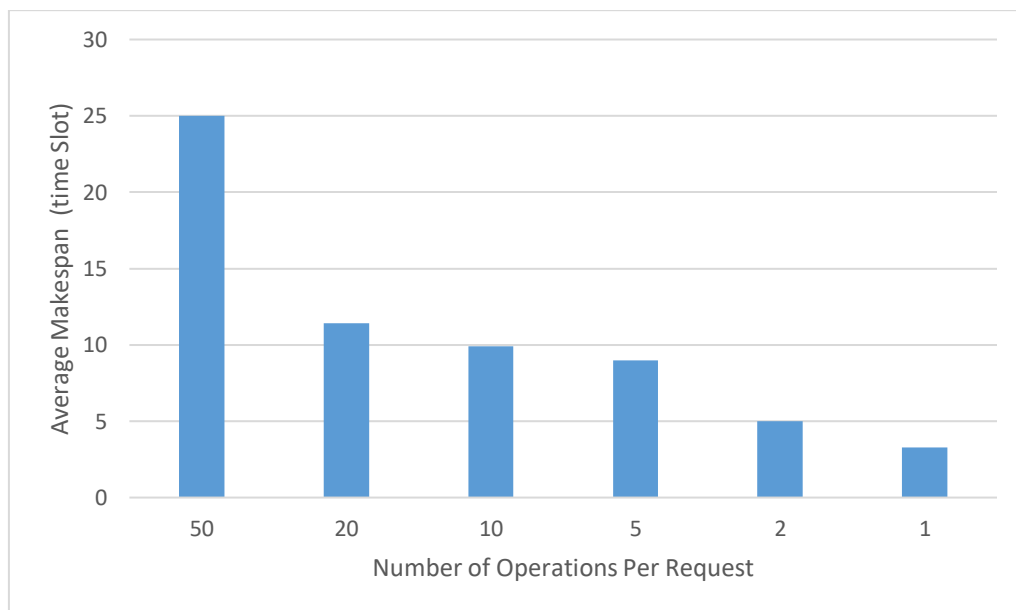


Figure 6.6: SABS average makespan by varying the number of requests

In the bar chart in Figure 6.6, the larger the number of operations per request, the higher the average makespan. Two reasons can clarify this observation: First, the average makespan is calculated as the sum of all requests' makespan divided by the number of all requests. Therefore, a smaller number of requests leads to a higher

average makespan. Second, there are dependencies among operations of the same request. However, there are no dependencies among different requests. Hence, the more operations per request, the more dependencies exist among them. Accordingly, the more the dependencies among operations the fewer the operations that can be executed in parallel. As a result, more operations are executed in a sequential fashion, consequentially increasing the average makespan. Furthermore, it is evident from Table 6.5 that the execution time is influenced by the number of requests in the scheduling problem. The relationship can be shown in the graph plotted in Figure 6.7.

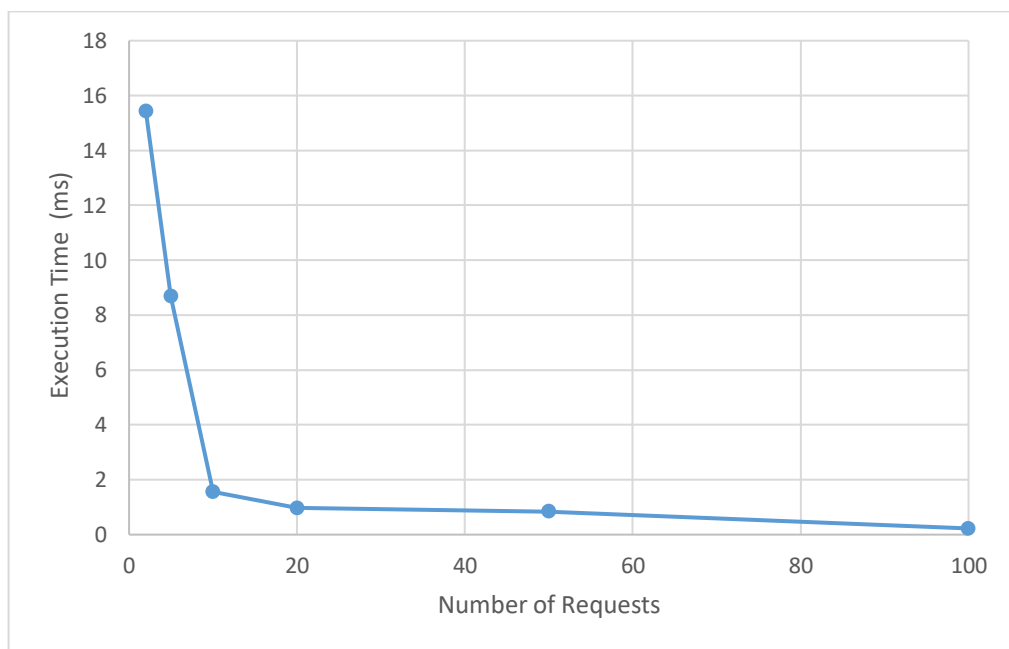


Figure 6.7: SABS execution time by varying the number of requests

In Figure 6.7, the execution time dramatically increases with the reduction of the number of requests. This observation can be explained with the following: the number of dependencies among operations increases as the number of requests decreases. Accordingly, more calculations are required to satisfy the dependency constraint. Thus, the scheduling complexity rises intensely, consequentially increasing the execution time of the algorithm.

Furthermore, the proposed algorithm performance is analyzed while varying the number of operations and keeping the number of requests fixed to further study the effect of the operation dependencies. The changes on the makespan and the execution

time of the proposed solution by varying the total number of operations and fixing all other parameters are logged in Table 6.6.

Table 6.6: The impact of varying the number of operations on SABS algorithm

Experiment No.	Number of Requests	Operations Per Request	Number of Operations	Number of Small Cells	Number of Compute Blocks	Average Makespan (time slot)	Execution Time (ms)
1	10	1	10	2	50	0.6	0.07
2	10	2	20	2	50	1.5	0.173
3	10	3	30	2	50	3.1	0.466
4	10	4	40	2	50	3.3	0.776
5	10	5	50	2	50	4.9	0.963
6	10	6	60	2	50	5.3	1.144
7	10	20	200	2	50	N/A	N/A

In Table 6.6, the average makespan is rising with the growth of the total number of operations. Moreover, experiment 7 illustrates the system behavior when the available resources are insufficient to satisfy request requirements. Accordingly, the system fails to schedule the requests within the given deadlines. Furthermore, the number of operations per request is recorded to examine the effect on the average makespan. In fact, there is a relationship between the number of operations per request and the average makespan. Such a relationship is illustrated in the bar chart shown in Figure 6.8.

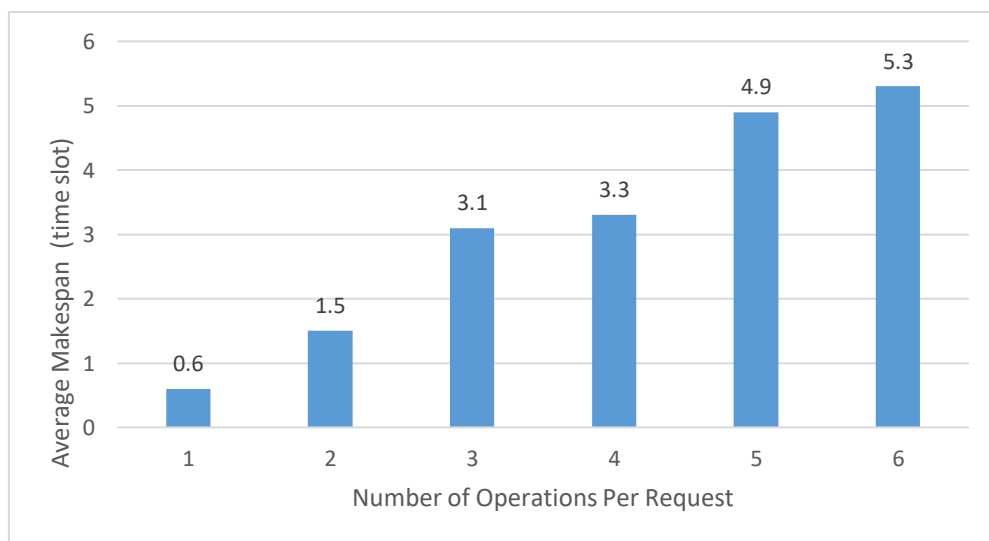


Figure 6.8: SABS average makespan vs. number of operations per request

In the chart shown in Figure 6.8, it is perceptible that the average makespan increases as the number of operations per request increases. This observation can also be explained as a result of the escalation in the number of dependencies among operations. This complies with the conclusion reached earlier while interpreting the effect of varying the number of requests on the average makespan, shown in Figure 6.6. Moreover, the proposed algorithm's execution time is influenced by the number of operations per request. To further examine the impact, a graph describes the relationship between the execution time and the number of operations per request is shown in Figure 6.9.

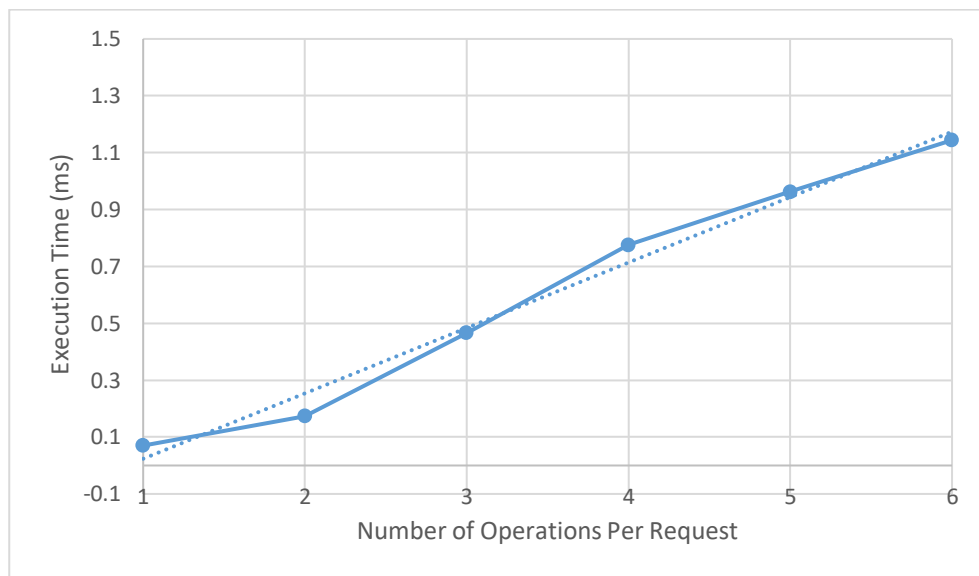


Figure 6.9: SABS execution time vs. number of operations per request

The graph in Figure 6.9 shows the impact of the number of operations per request on the execution time of the proposed SABS algorithm. It is noticeable that rising the number of operations per request increases the execution time of the algorithm. This also agrees with the conclusion inferred earlier while examining the influence of changing the number of requests on the execution time, demonstrated in Figure 6.7. Furthermore, in Figure 6.9, the complexity instigated by increasing the number of operations per request is approximately linear in time.

Thereafter, sensitivity analysis is performed for the parameters that represent system resources, small cells and their compute blocks. Considering different numbers

of small cells, the average makespan and the execution time of the proposed algorithm are noted in Table 6.7.

Table 6.7: The impact of varying the number of small cells on SABS algorithm

Experiment No.	Number of Requests	Number of Operations	Number of Small Cells	Compute Blocks Per Small Cell	Number of Compute Blocks	Average Makespan (time slot)	Execution Time (ms)
1	10	100	1	60	60	7.4	4.267
2	10	100	2	30	60	7.5	4.185
3	10	100	3	20	60	7.7	4.42
4	10	100	4	15	60	8.6	4.456
5	10	100	5	12	60	8.2	4.965
6	10	100	6	10	60	8.9	4.149

The relationship between the number of small cells and the average makespan is summarized in Figure 6.10.

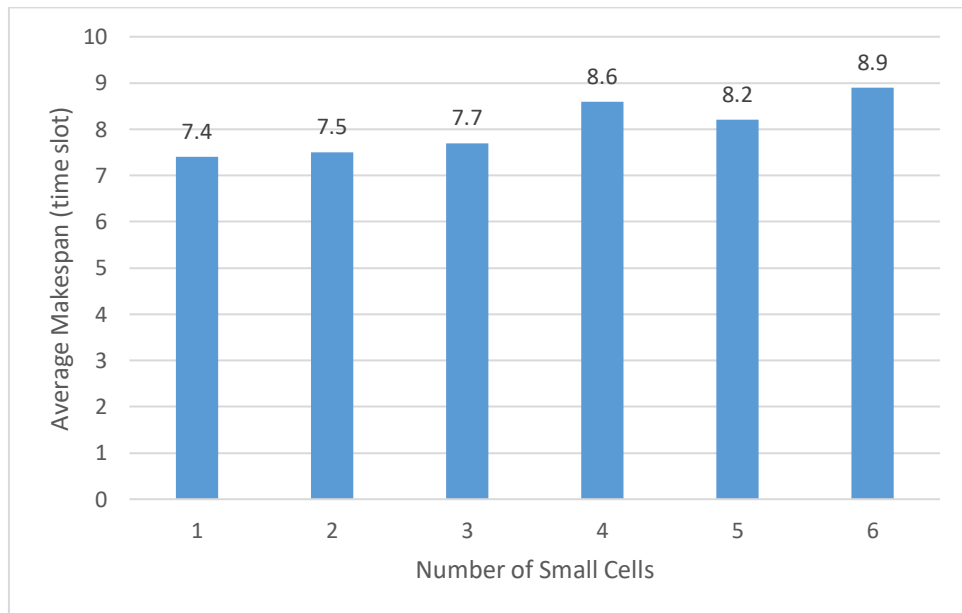


Figure 6.10: SABS average makespan vs. the number of small cells

In the bar chart in Figure 6.10, the average makespan slightly increases by raising the number of small cells given the same number of compute blocks. The cause of such results is the indivisibility constraint, equation (11), which prevents any operation from being executed by multiple small cells. In addition, scattering the

compute blocks across more small cells decreases the number of compute blocks per small cell. As a result, fewer operations can be executed in parallel which eventually leads to a larger average makespan. Moreover, the correlation between the proposed algorithm execution time and the number of small cells is illustrated in the graph shown in Figure 6.11.

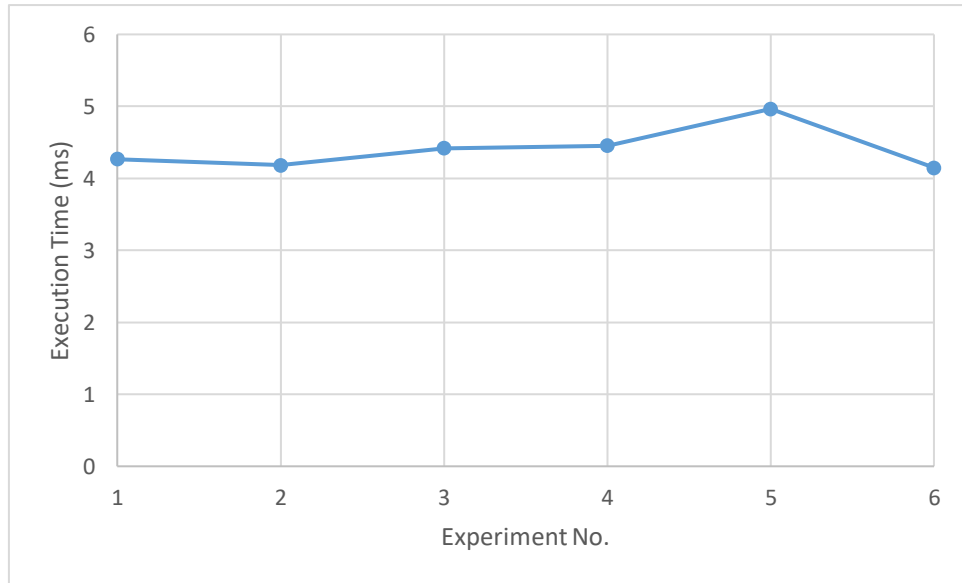


Figure 6.11: SABS execution time with different numbers of small cells

In the graph in Figure 6.11, the execution time of the proposed SABS algorithm seems to be unaffected by the number of small cells. Despite the incrementation in the number of small cells, the number of compute blocks is unchanged throughout the experiments. Therefore, the search space, from which a solution is obtained, remains the same. Thus, the uncorrelation between the number of small cells and the execution time of the algorithm is reasonable. Moreover, changing the number of compute blocks per small cell has no impact on performance as the number varies from 60 to 10 with no remarkable change in the execution time. Furthermore, to examine the outcome of increasing the search space, more experiments considering different numbers of compute blocks are performed and the resultant average makespan and execution time of the proposed algorithm are logged. The experimentation parameters and results are listed in Table 6.8.

Table 6.8: The impact of varying the number of compute blocks on SABS algorithm

Experiment No.	Number of Requests	Number of Operations	Number of Small Cells	Compute Blocks Per Small Cell	Number of Compute Blocks	Average Makespan (time slot)	Execution Time (ms)
1	10	100	2	10	20	26.5	3.75
2	10	100	2	15	30	13.5	3.303
3	10	100	2	20	40	10.9	2.155
4	10	100	2	25	50	9.1	1.657
5	10	100	2	30	60	8.6	1.2395
6	10	100	2	35	70	7.2	1.108

In Table 6.8, by increasing the total number of compute blocks from 10 to 70, the average makespan falls from 26.5 time slots to 7.2 time slots while the execution time drops from 3.75 ms to 1.108 ms. To further examine the relationship between the total number of compute blocks and the average makespan, a graph describes the correlation between them is illustrated in Figure 6.12.

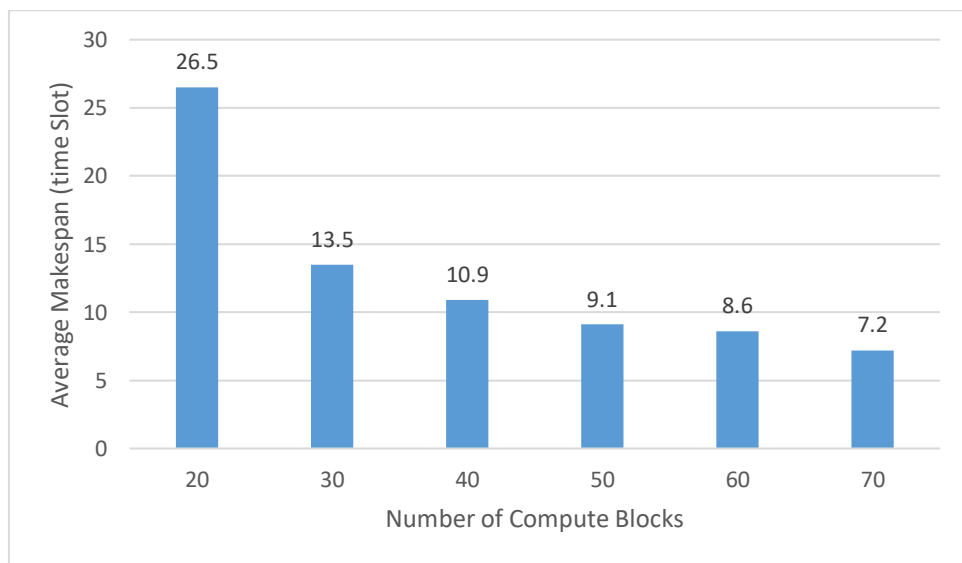


Figure 6.12: SABS average makespan vs. the number of compute blocks

The graph in Figure 6.12 shows that the more the available compute blocks, the smaller the average makespan. The reason is that there are more available resources to process more requests. Therefore, more operations can be parallelly executed at the same time, leading to a lower average makespan. Additionally, it is visible in Figure 6.12 that the effect on the average makespan decreases as more compute blocks are added to the problem. For instance, the average makespan dropped by 13 time slots

after incrementing the number of compute blocks by 10, from 20 to 30. Afterward, the same number of compute blocks is added to make it 40. However, the average makespan falls only 2.6 time slots. This can be explained as follows: although increasing the number of compute blocks enables more operations to be executed at the same time, there is a point by which the dependencies among operations prevent more parallel executions. Therefore, the change in the average makespan will eventually saturate. Moreover, it is noteworthy that the effect on the average makespan, shown in Figure 6.12, is mainly caused by the total number of compute blocks and not the number of compute blocks per small cell. The conclusion obtained from Table 6.7 confirms the uncorrelation between the average makespan and the number of compute blocks per small cell. Furthermore, the effect of having more resources on the execution time is described by the graph conducted in Figure 6.13.

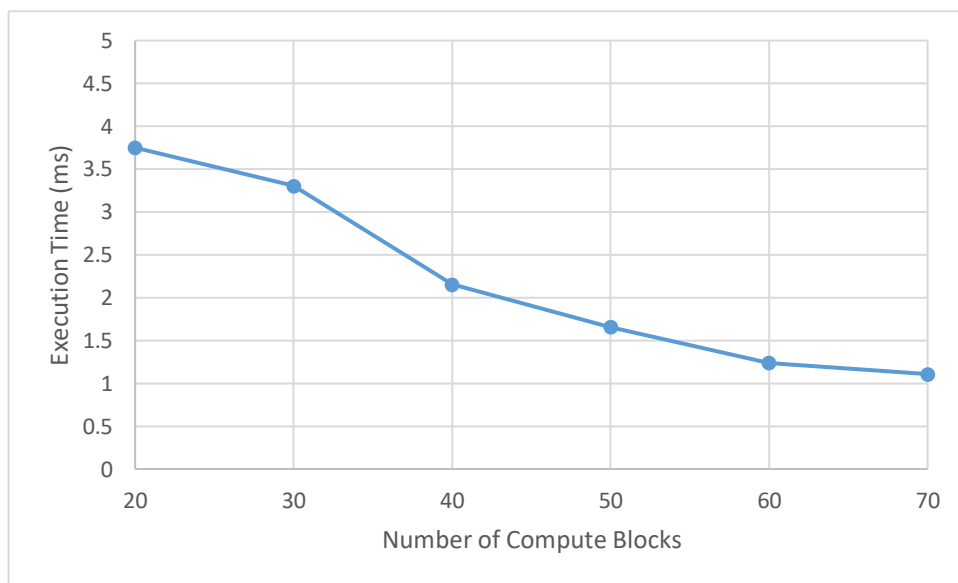


Figure 6.13: SABS execution time vs. the number of compute blocks

In Figure 6.13, the graph shows that the execution time drops significantly by increasing the number of compute blocks. The design of the proposed SABS algorithm is the origin of such results. In specific, the SABS algorithm starts with the most parallelized solution, with the least degree of sequence. If the number of compute blocks is limited, the initial schedule will have several small idle resource blocks. Thus, the search space to reach the convergence state increases, leading to greater execution

time. On the other hand, the greater the number of compute blocks, the closer the initial solution is to the final best solution, therefore reducing the number of iterations needed for the algorithm to converge, eventually requiring less execution time. The number of compute blocks per small cell is not the influential factor on the execution time as the conclusion acquired from Table 6.7 also confirms the uncorrelation.

To conclude, the performance of the proposed SABS algorithm is adequate compared to the optimal solutions obtained by the CPLEX optimization tool. In general, there is an impact of increasing the problem size on the algorithm execution time and the resulting average makespan. In specific, increasing the total number of operations as well as the number of operations per request increases both the average makespan and the execution time. On the other hand, the greater the number of compute blocks, the smaller the average makespan and the less the execution time. Finally, the number of small cells has an influence on the average makespan in a directly proportional manner. Nevertheless, it has no impact on the algorithm execution time.

6.3. Decentralized Approach Experimentations

In this section, multiple experiments are presented to evaluate the performance of the proposed decentralized approach. Different numbers of resources and operations are considered to examine the overall system performance. Furthermore, sensitivity analysis is performed to address the impact of each parameter on the system behavior. All experimentations are conducted on a machine that consists of Intel(R) Xeon(R) Gold 5118 processor with a speed of 2.3 GHz and a RAM of 48 GB. The 64-bit version of Microsoft Windows 10 Education is the operating system installed.

6.3.1. CPLEX simulation results. The proposed decentralized framework and all the associated optimization models are implemented in OPL and solved using CP engine provided by the IBM ILOG CPLEX optimization studio. Moreover, the proposed interaction structure, presented in Figure 4.1, is implemented in JavaScript. The parameters used in the simulation process are the same parameters used in the centralized approach experimentations, shown in Table 3.3. Using the developed data generator, different values of system parameters are considered. Table 6.9 shows information regarding the experimentations simulated using CPLEX.

Table 6.9: CPLEX experiments for the decentralized approach

Experiment No.	Number of Requests	Number of Operations	Number of Small Cells	Number of Providers	Number of Consumers	Number of Compute Blocks
1	8	31	3	2	1	17
2	15	59	6	4	2	34
3	23	90	9	6	3	51
4	30	118	12	8	4	68

For each experiment in Table 6.9, the scheduling problem is solved using CPLEX optimization tool to get an optimal solution. Accordingly, the average makespan and the execution time are logged and listed in Table 6.10.

Table 6.10: CPLEX results for the decentralized approach

Experiment No.	Average Makespan (time slots)	Execution Time (s)
1	5.125	550
2	4.5	593
3	4.826	10211
4	N/A	N/A

To examine the complexity of the decentralized scheduling problem, Figure 6.14 illustrates a bar chart that shows the impact of each experiment on the execution time.

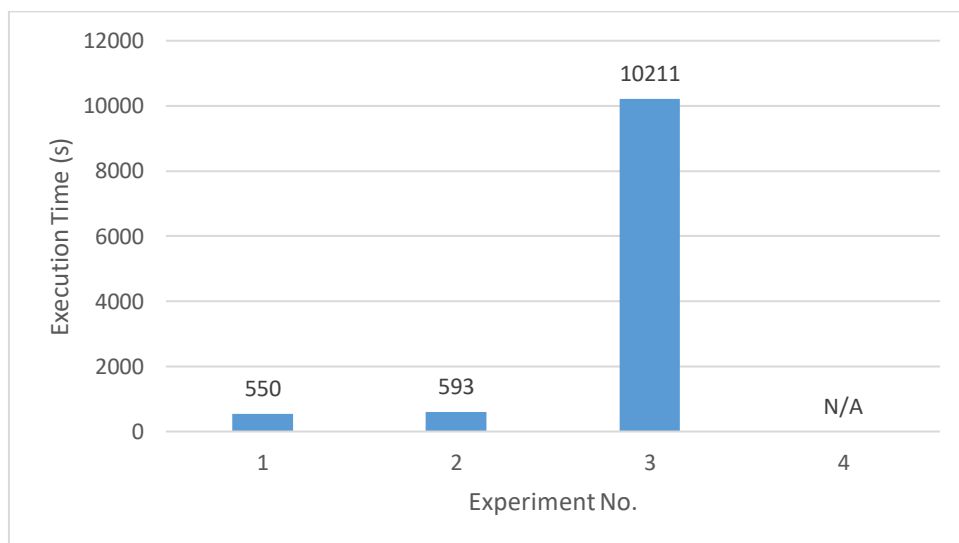


Figure 6.14: Decentralized approach execution time (CPLEX)

In the bar chart in Figure 6.14, it is evident that the problem size affects the execution time dramatically. This complies with the fact that the decentralized scheduling problem is an NP-Hard problem. Furthermore, the influence on the average makespan is shown in the bar chart presented in Figure 6.15.

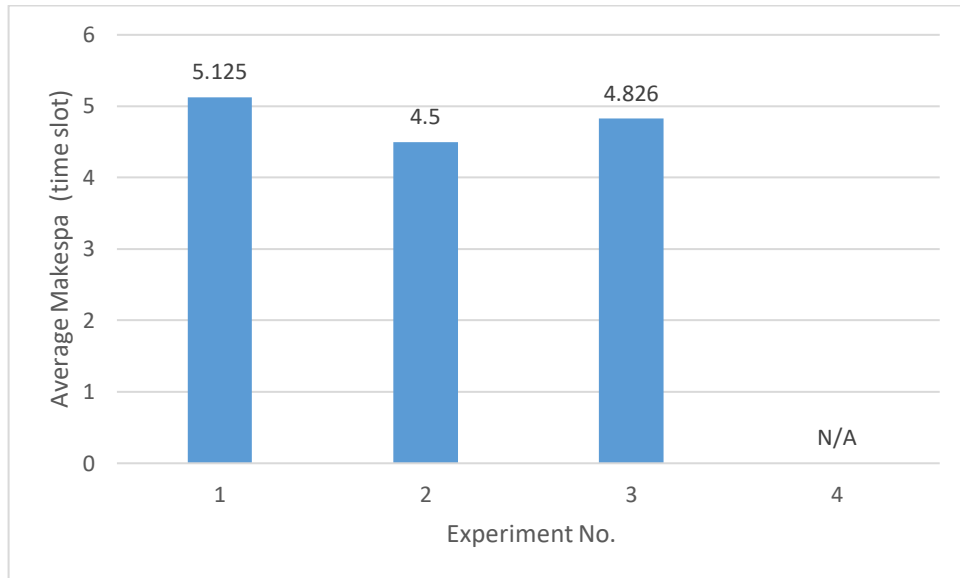


Figure 6.15: Decentralized approach average makespan (CPLEX)

In Figure 6.15, the average makespan fluctuates by increasing the problem size. Accordingly, the result does not provide enough information to reach a certain deduction. Therefore, later in this section, sensitivity analysis is performed to further understand the influence of each parameter on the overall system performance.

6.3.2. Decentralized auction-based approach performance evaluation. The proposed decentralized auction-based heuristic is implemented in C++. In this subsection, the proposed solution is simulated, and the results are presented. In addition, the optimal solutions obtained by CPLEX is used as a benchmark to evaluate the quality of the proposed solution. For the sake of the comparison, the generated data in Table 6.9 and the parameters in Table 3.3 are used. The results of running the simulation of the proposed solution are presented in Table 6.11. To compare the average makespan in Table 6.11 with the corresponding optimal values, a bar chart showing the average makespan obtained from the decentralized heuristic and CPLEX is depicted in Figure 6.16.

Table 6.11: The decentralized heuristic average makespan and execution time

Experiment No.	Average Makespan (time slot)	Execution Time (ms)
1	5.5	0.67
2	4.73	0.685
3	5	0.671
4	4.73	0.731

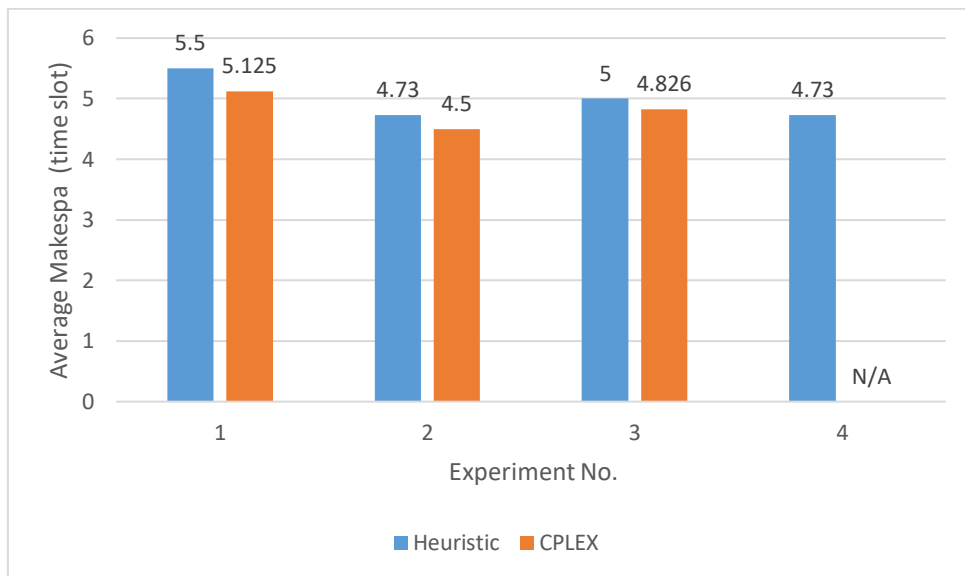


Figure 6.16: Decentralized heuristic vs. CPLEX (average makespan)

The average makespan acquired by the proposed decentralized heuristic is very close to the optimal average makespan obtained by CPLEX. Therefore, these results are acceptable considering the amount of time saved in solving the scheduling problem. Moreover, to assess the quality of the proposed algorithm, the proposed heuristic algorithms for the decentralized approach and the centralized approach are simulated and compared considering various problem sizes. Experimentation parameters are listed in Table 6.12.

For the centralized approach, all the experiments in Table 6.12 are conducted using the proposed SABS algorithm. On the other hand, the proposed decentralized auction-based approach includes both SABS and ABWD algorithms. The resultant average makespan and the execution time are depicted in Table 6.13.

In Table 6.13, the average makespan is unaffected by the problem size as opposed to the execution time because more operations are executed by more resources.

Figure 6.17 shows the average makespan and its moving average for both considered approaches.

Table 6.12: Decentralized heuristic experimentation parameters

Experiment No.	Number of Requests	Number of Operations	Number of Small Cells	Number of Providers	Number of Consumers	Number of Compute Blocks
1	8	31	3	2	1	17
2	10	39	4	3	1	23
3	14	55	5	3	2	28
4	15	59	6	4	2	34
5	17	67	7	5	2	40
6	22	86	8	5	3	45
7	23	90	9	6	3	51
8	25	98	10	7	3	57
9	29	144	11	7	4	62
10	30	118	12	8	4	68
11	32	126	13	9	4	74
12	37	145	14	9	5	79
13	38	149	15	10	5	85

Table 6.13: Experimental results: centralized vs. decentralized approach

Experiment No.	Decentralized Heuristic		Centralized Heuristic	
	Average Makespan (time slot)	Execution Time (ms)	Average Makespan (time slot)	Execution Time (ms)
1	5.5	0.67	5.375	0.572
2	5.3	0.672	5	0.957
3	4.66667	0.67	5.866666667	1.123
4	4.5625	0.685	5.1875	2.18
5	4.55556	0.665	5.166666667	2.073
6	4.95652	0.875	5.652173913	2.377
7	4.875	0.671	5.833333333	1.474
8	4.84615	0.689	5.192307692	4.235
9	4.6129	0.884	5.516129032	5.457
10	4.5625	0.731	5.34375	6.613
11	4.5588	0.708	5.323529412	8.086
12	4.79487	0.85	5.512820513	13.055
13	4.75	0.674	5.25	13.8

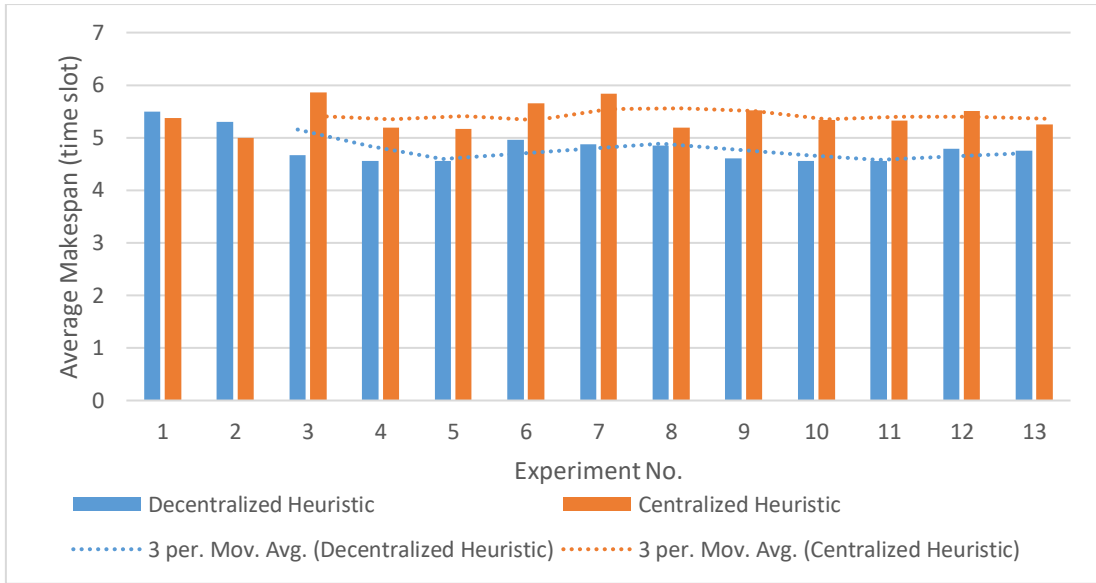


Figure 6.17: Centralized heuristic vs decentralized heuristic (average makespan)

In Figure 6.17, the performance of the decentralized heuristic overcomes the centralized SABS algorithm. This is because the scheduling problem is divided into smaller subproblems which are dispersed across multiple small cells in the decentralized approach. Accordingly, each small cell handles its simpler scheduling problem. Since SABS quality is influenced by the problem size, having the problem divided across multiple agents improves the results. Moreover, in Table 6.13, it is noticeable that the decentralized heuristic execution time is independent of the problem size as opposed to the centralized heuristic execution time. For further inspection, the change in the execution time of the two heuristics is plotted in Figure 6.18.

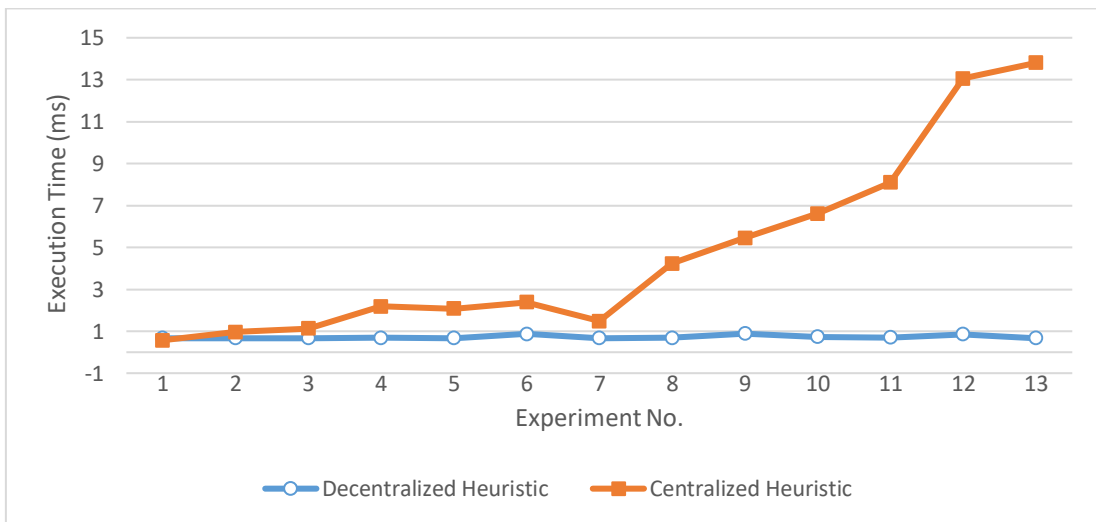


Figure 6.18: Centralized heuristic vs. decentralized heuristic (execution time)

In Figure 6.18, the execution time of the decentralized auction-based heuristic is unaffected by the problem size. However, as deduced earlier, the complexity of the centralized heuristic, SABS, increases with the growth of the problem size. Therefore, this confirms the scalability of the decentralized auction-based heuristic solution.

6.3.3. Decentralized auction-based heuristic sensitivity analysis. In this subsection, the impact of each system parameter on the average makespan and the execution time of the proposed decentralized heuristic solution is investigated. Sensitivity analysis is performed by varying the value of each input parameter individually. The decentralized proposed solution utilizes two algorithms, namely ABWD and SABS. In subsection 6.2.3, sensitivity analysis for the SABS algorithm was presented. Therefore, this subsection focuses on the sensitivity analysis for the ABWD algorithm. The inputs for the proposed ABWD algorithm are the number of consumer agents and the number of provider agents. Hence, the impact of the two input parameters is to be investigated. First, the number of provider agents is systematically changed while keeping the number of consumer agents fixed. Experiment parameters and results are shown in Table 6.14.

Table 6.14: The impact of the number of provider agents on ABWD algorithm

Experiment No.	Number of Requests	Number of Operations	Number of Providers	Number of Consumers	Average Makespan (time slot)	Execution Time (ms)
1	30	115	5	5	5.6667	0.7
2	31	119	6	5	5.67742	0.743
3	32	123	7	5	5.53124	0.704
4	33	127	8	5	5.39394	0.675
5	34	131	9	5	5.26471	0.696
6	35	135	10	5	5.14286	0.735

In Table 6.14, the results show that the average makespan drops as the number of provider agents increases. To further examine the relationship, a bar chart showing the makespan corresponding to the number of provider agents is presented in Figure 6.19.

In Figure 6.19, the larger the number of provider agents, the smaller the average makespan. The reason behind this drop is that more providers leads to more available

shareable resources. Therefore, more requests can be executed in parallel. Moreover, the change in the makespan and the execution time of the proposed heuristic solution is examined by varying the number of consumers. The results and the parameters of the experiments are listed in Table 6.15.

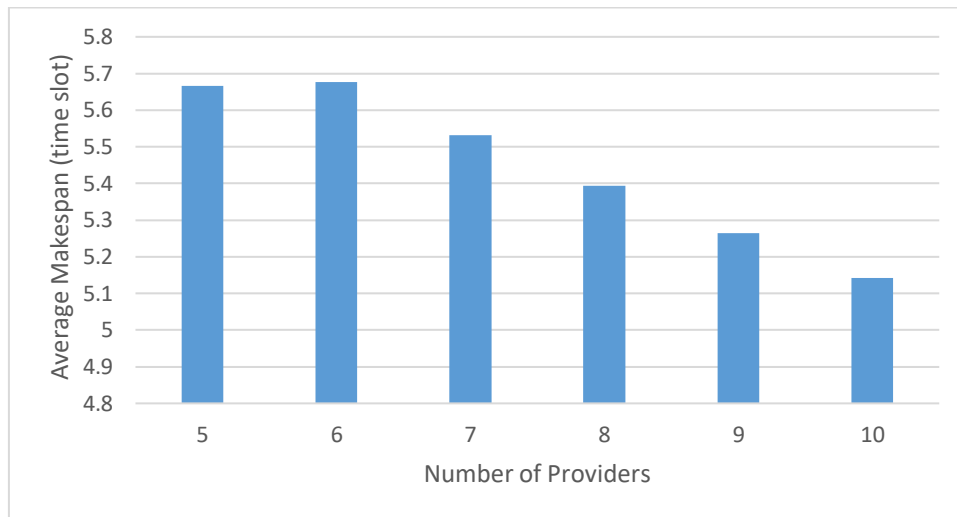


Figure 6.19: Average makespan by varying the number of provider agents

Table 6.15: The impact of the number of consumer agents on ABWD algorithm

Experiment No.	Number of Requests	Number of Operations	Number of Providers	Number of Consumers	Average Makespan (time slot)	Execution Time (ms)
1	35	135	10	5	5.14286	0.735
2	40	154	10	6	5.35	0.697
3	45	173	10	7	5.511111	0.661
4	50	192	10	8	5.64	0.703
5	55	211	10	9	5.74545	0.773
6	60	230	10	10	5.6667	0.76
7	65	249	10	11	N/A	N/A

Table 6.15 shows that the average makespan rises with the growth of the number of consumer agents. To inspect the relationship, a bar chart presenting the average makespan corresponding to different numbers of consumer agents is illustrated in Figure 6.20.

In Figure 6.20, the larger the number of consumer agents, the larger the average makespan. Such a correlation exists because the more the number of consumer agents, the more the number of handed-over requests. Therefore, more shareable available

resources are required. Since the number of providers is fixed, the number of shareable available resources is also unchanged. Thus, the additional handed-over requests will be scheduled in a sequential manner, increasing the average makespan. Eventually, the shareable available resources will be insufficient to accommodate all the handed-over requests, leading to system failure, noticeable in experiment 7. Finally, the difference in the execution time of the proposed decentralized solution is examined while varying the number of consumer and provider agents. A graph, extracted from Table 6.14 and 6.15, showing the execution time of the proposed heuristic is depicted in Figure 6.21.

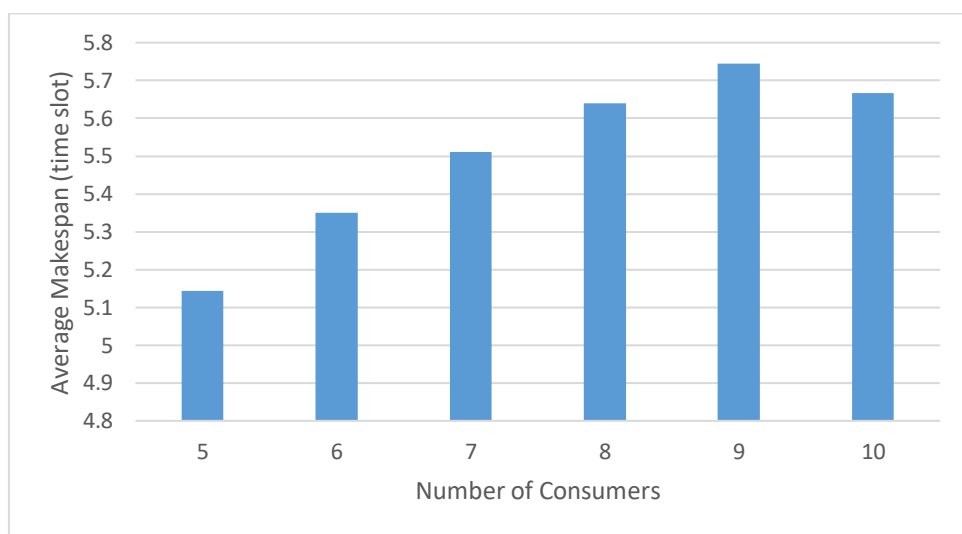


Figure 6.20: Average makespan by varying the number of consumer agents

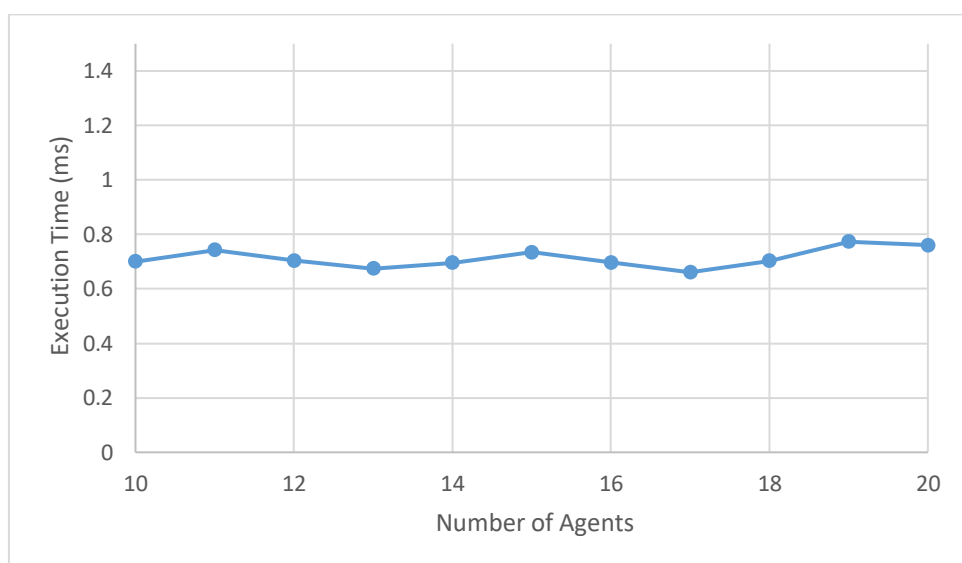


Figure 6.21: Execution time vs. number of agents

The graph in Figure 6.21 combines the experiments in Table 6.14 and Table 6.15. The graph in Figure 6.21 shows an approximately constant execution time with an average of 726 μ s. Consequentially, the number of agents, whether providers or consumers, has no influence on the execution time of the proposed algorithm. This conforms with what is concluded in subsection 6.3.2 that the system is scalable.

To summarize, the quality of the decentralized heuristic algorithm is tested by comparing the average makespan with the results obtained by using CPLEX optimization engine. The resultant values are very close to the optimal values. Furthermore, the performance of the decentralized auction-based heuristic solution is assessed by comparing it to the centralized SABS algorithm. The average makespan of the decentralized heuristic is lower than the centralized heuristic. Moreover, the execution time of the decentralized heuristic remained approximately constant with different problem sizes as opposed to the centralized algorithm. Therefore, we can conclude that the proposed decentralized auction-based solution is adequate in solving scheduling problems in scaled sizes, consequentially able to accommodate the momentous upsurge of connected user-devices.

Chapter 7. Case Study: Chronic Obstructive Pulmonary Disease

Chronic Obstructive Pulmonary Disease (COPD) is a term that describes multiple progressive lung disorders. A patient diagnosed with this disease suffers from an unbalanced amount of exerted oxygen, leading to breathing difficulties. COPD is a chronic, incurable, disease that needs continuous treatment. Therefore, patients need 24/7 monitoring and supervision to provide the needed amount of oxygen and maintain its healthy level. In this chapter, a case study about utilizing fog computing in COPD treatment is presented.

7.1. Fog Computing in Healthcare

Nowadays, traditional healthcare is found to be inconvenient for multiple reasons. For instance, physicians do not have the time capacity to offer a fair amount of time for diagnosing each patient. Furthermore, the lack of modern monitoring equipment led to the need for patient hospitalization which helps diseases to spread more. In addition, more medical staff are needed since the demand for healthcare is increasing due to aging, spreading of diseases and population growth. Furthermore, visiting a health facility for medical check-ups is costly, and it is predicted to be increasing in the future. Therefore, the community needs patient-centered care (PCC) alternative solution that exploits today's available technologies instead of traditional healthcare, clinic-centered solution [42]. To enable patient-centered healthcare, IoT-enabled technologies are used. By developing a domain-specific software, IoT smart devices, with the help of sensors, can remotely monitor the patients and send collected data accordingly. Subsequently, the data is processed to acquire useful information. Afterward, the appropriate healthcare worker, i.e. doctor, nurse or physician, is informed with the resultant knowledge for further decision making and diagnostic purposes. In some cases, an IoT smart device can take its own decision to provide the patient with the needed therapy. Such applications involve real-time requests with hard deadlines. In general, IoT technologies rely on cloud computing. However, the delay results from using cloud services makes the cloud paradigm unreliable in the healthcare domain. Thus, time-critical requests cannot be resolved by the cloud. To solve this problem, fog computing is considered to be a candidate. In addition, using fog computing in healthcare increases patient mobility, improves reliability and ensures continuous health monitoring [43].

7.2. Breath Assistance System

COPD patients suffer from issues in breathing due to the instability of the oxygen level supplied. Machines and devices that contain oxygen can be used to provide the needed amount of oxygen to the patients. However, the prescribed level of oxygen depends on specialists' assumption of oxygen minimal exertion. The assumption is relatively accurate if the patient is under monitoring in a hospital with no major activities. In other words, the patient's situation is characterized to be static. Therefore, the amount of oxygen prescribed is fixed. In this static case, a device can periodically supply the needed amount of oxygen as prescribed by doctors. The static problem is actually investigated before in [44]. Nevertheless, the static situation does not consider any physical activity although exercises, even basic ones, are very useful and beneficial to COPD patients [45]. The most basic exercise for COPD patients is walking. Moreover, walking-based therapy is proven to enhance the treatment and life quality of COPD patients [46]. Hence, it is important to consider and study a breathing assistance system for COPD patients that considers the dynamic nature of oxygen needs. Dynamic monitoring and assistance in a fog computing environment are investigated in [47]. The work describes the nature of such service requests in the environment of fog computing. The environment of such an architecture, considering fog computing and 5G infrastructure, can be shown in Figure 7.1.

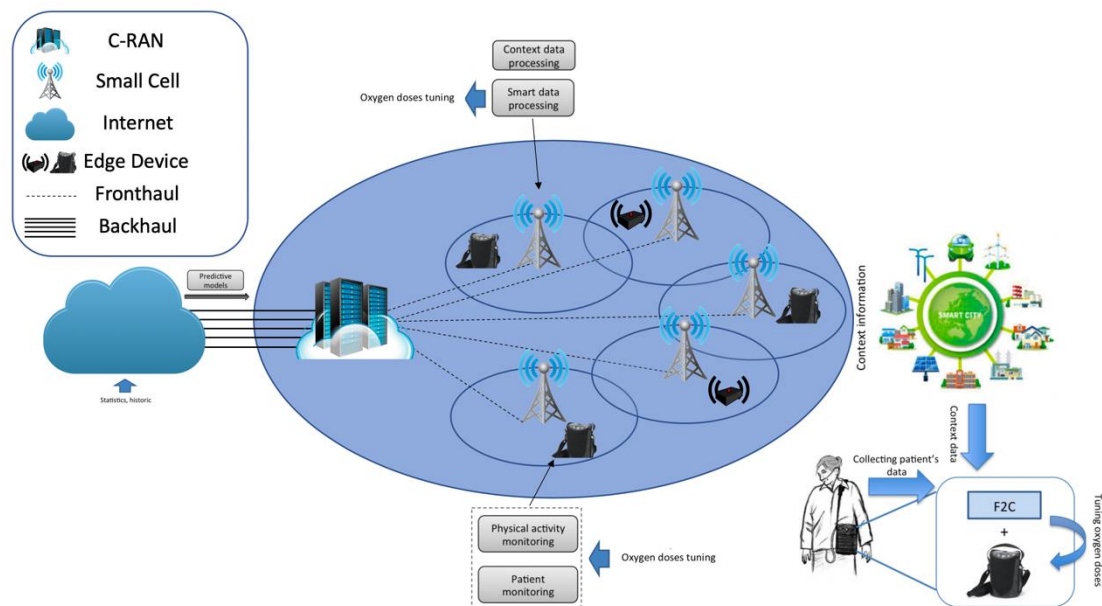


Figure 7.1: Fog/5G environment of the breath assistance system

7.2.1. Oxygen adjustment as a service request. As described in [47], monitoring a patient and providing a suitable amount of oxygen is a real-time service request. Such a critical request needs to be evaluated by the fog computing layer to avoid the delay of cloud datacenters. Moreover, in addition to patients' conditions and their physical activities, the actual amount of oxygen required depends on other external factors such as air pollution. Information about external factors is periodically sent to the cloud for data analytics. The result is a predictive model, sent back to the fog nodes, that estimates the required amount of oxygen needed.

The portable breath assistance device monitors the patient physical activity and arterial oxygen partial pressure. Afterward, the device sends the related information to the fog layer periodically. Accordingly, the data is encoded in a suitable format to be ready for sending. As a fog node receives the data, it decodes the data to be in a ready-to-read format. Furthermore, the predictive model for oxygen level prediction is loaded and applied. Consequentially, the fog node decides whether to supply oxygen to the patient or only notify the patient about a potential need. Finally, the resultant data is encoded to be sent to the device that takes action accordingly. The operations of the oxygen adjustment request can be summarized in Table 7.1.

Table 7.1: Operations of a service request (oxygen adjustment)

Operation No.	Description	Instruction Count (MI)	Dependency
1	Data Decoding	0.2	N/A
2	Loading the predictive model	0.3	N/A
3	Applies the model and Predict the output	0.4	1,2
4	Data Encoding	0.3	3

7.2.2. Simulation Results. For this simulation, an environment that consists of 2 small cells is considered. The first one has 4 compute blocks and the second one has 5 compute blocks. Each compute block has the capability of 150 MIPS. In this environment, there are three patients that are connected to the fog layer through their breath assistance portable devices. Each one demands a service request of adjusting the oxygen level. Therefore, according to Table 7.1, the total number of operations to be scheduled is 12. As a result, this scheduling problem instance is solved using the

proposed heuristic as well as the CPLEX solver. The proposed algorithm managed to achieve an average makespan of 4.33 time slots while CPLEX acquired the optimal schedule with an average makespan of 4 time slots. Finally, the algorithm result is good enough compared to CPLEX since the execution time between them is tremendous. CPLEX obtained the optimal schedule in 10.20 s while the proposed heuristic scheduled all operations in 361 μ s.

Chapter 8. Conclusion and Future Work

This work focused on the problem of minimizing the latency of fog computing within the 5G network. The problem of minimizing the latency was defined as a scheduling problem that involves allocating small cell resources to service requests received from user-devices in the 5G network. By considering the environment and its characteristics, the scheduling problem was modeled as an optimization problem. The proposed optimization model represents the scheduling problem given full knowledge of the resources and requests within the network. However, a top-view solution is not adequate given the decentralized nature of the small cells in 5G networks. Furthermore, this decentralized environment led to interdependencies between the small cells in the physical setup as well as the mental domain. Therefore, a decentralized framework is proposed to solve the interdependency problem. The proposed framework consists of multiple subproblems, namely the self-classification, local scheduling and winner determination problems. Each subproblem is formulated as a different optimization model. Moreover, a game-theoretic approach is used to map the winner determination problem into a non-cooperative simultaneous combinatorial game with complete but imperfect information. Afterward, an economic-based approach is considered to model the game as a combinatorial auction problem. Furthermore, agent preferences are represented as utility values that reflect their local interests. Subsequently, all the proposed optimization models are validated by solving the models by hand and by using an optimization engine where results were shown to be the same. Solving a small scheduling problem instance of 16 operations and 20 available compute blocks using CPLEX required 2 hours, 19 minutes and 10 seconds of execution time. Moreover, the process of scheduling 38 operations in 22 compute blocks failed to complete after running the simulation for one week. In the validation process of the proposed decentralized framework, solving a small scheduling problem instance of 90 operations and 51 available compute blocks using CPLEX required 2 hours, 50 minutes and 11 seconds of execution time. Furthermore, after running the simulation for one week, the solver failed to schedule 118 operations in 68 compute blocks.

The problem of scheduling service requests across small cell resources is involved in the self-classification and local scheduling processes. The execution time

of the CPLEX solver is unrealistic due to the NP-hardness of the scheduling problem. Hence, a simulated annealing-based scheduling algorithm is proposed to solve the scheduling problem in polynomial time. The SABS algorithm is implemented and tested in C++ programming language. The results of the algorithm were compared to the optimal solutions acquired by an optimization engine. The comparison proved the adequacy of the proposed SABS algorithm. Furthermore, sensitivity analysis is presented to address the impact of each input parameter on the proposed SABS algorithm performance. Accordingly, it is found that increasing the total number of operations as well as the number of operations per request, increases the average makespan and the execution time of the proposed SABS algorithm. Moreover, the outcomes confirmed that increasing the number of operations eventually leads to system failure due to the insufficiency of resources to satisfy request requirements. Additionally, experiments demonstrated that the greater the number of compute blocks, the smaller the average makespan and the less the execution time. Moreover, the number of small cells affects the average makespan in a directly proportional manner, nevertheless, it has no influence on the execution time of the proposed algorithm.

Furthermore, the winner determination problem is formulated as a combinatorial auction problem. Since the combinatorial auction problem is an NP-hard problem, an auction-based winner determination algorithm is proposed to solve the problem in polynomial time. Both SABS and ABWD algorithms are part of the proposed decentralized heuristic solution. The proposed heuristic approach is implemented in C++ programming language. Its performance is verified with the optimal solutions acquired by an optimization engine. The results were found to be close to the optimal solutions, consequentially showing the validity of the proposed heuristic solution. Moreover, a centralized approach is considered and is used as a benchmark for the proposed decentralized heuristic solution. The outcomes showed the competence of the proposed decentralized heuristic in terms of average makespan and execution time. In addition, the execution time of the proposed decentralized heuristic remained approximately constant, with an average of 726 μ s, considering different problem sizes as opposed to the centralized approach. Accordingly, the proposed decentralized solution is found to be scalable, therefore able to manage the gigantic rise in connected user-devices. Finally, sensitivity analysis is conducted to recognize the influence of the number of consumer agents and the number of provider

agents on the proposed ABWD algorithm performance. According to the analysis, the average makespan is directly proportional to the number of provider agents but inversely proportional to the number of consumer agents. Nonetheless, the algorithm's execution time is found to be independent of the number of agents.

In future works, the performance of the proposed decentralized solution can be further investigated by implementing the decentralized framework in physical small cells. Accordingly, a testbed is to be built that consists of multiple small cells in which the proposed SABS algorithm shall be implemented for the purpose of obtaining practical results. In addition, small cells shall be connected to a processing unit to be employed as the coordinator. Furthermore, the proposed framework shall be extended to consider opportunistic user-devices, such as mobile devices, as part of the system to provide more computational resources. In such a system, an incentive-based framework shall be used to motivate user-devices to share their computational resources to execute service requests generated by other users within the 5G network.

References

- [1] M. Agiwal, A. Roy, and N. Saxena, “Next Generation 5G Wireless Networks: A Comprehensive Survey,” *IEEE Communication Survey Tutorials*, vol. 18, no. 3, pp. 1617–1655, 2016.
- [2] Cisco Systems, “Fog Computing and the Internet of Things: Extend the Cloud to where the things are,” *Cisco Systems*, 2015. [Online]. Available: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf. [Accessed: May 21, 2019].
- [3] E. Wikström and U. M. Emilsson, “Autonomy and Control in Everyday Life in Care of Older People in Nursing Homes,” *Journal of Housing For the Elderly*, vol. 28, no. 1, pp. 41–62, 2014.
- [4] S. Sarkar, S. Chatterjee, and S. Misra, “Assessment of the Suitability of Fog Computing in the Context of Internet of Things,” *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 46–59, 2018.
- [5] J. He, J. Wei, K. Chen, Z. Tang, Y. Zhou, and Y. Zhang, “Multitier Fog Computing With Large-Scale IoT Data Analytics for Smart Cities,” *IEEE Internet Things J.*, vol. 5, no. 2, pp. 677–686, 2018.
- [6] “Etisalat plans 5G coverage throughout UAE this year,” Feb. 17, 2019. [Online]. Available: <https://www.commsmea.com/technology/18926-etisalat-plans-5g-coverage-throughout-uae-this-year>. [Accessed: May 9, 2019].
- [7] A. Pratap, R. Misra, and S. K. Das, “Resource allocation to maximize fairness and minimize interference for maximum spectrum reuse in 5G cellular networks,” in *2018 IEEE 19th International Symposium on A World of Wireless, Mobile and Multimedia Networks*, Chania, 2018, pp. 1–9.
- [8] China Mobile Research Institute, “C-RAN: The Road Towards Green RAN,” *China Mobile Research Institute*, Oct. 2011. [Online]. Available: <https://pdfs.semanticscholar.org/ea3/ca62c9d5653e4f2318aed9ddb8992a505d3c.pdf>. [Accessed: May 9, 2019].
- [9] N. Cvijetic, “Optical network evolution for 5G mobile applications and SDN-based control,” in *2014 16th International Telecommunications Network Strategy and Planning Symposium*, Funchal, 2014, pp. 1–5.
- [10] Cisco Systems, “Cisco Small Cell Architecture.” *Cisco Systems*, May 2013. [Online]. Available: https://www.cisco.com/c/dam/global/hr_hr/assets/ciscoconnect/2013/pdfs/Cisco_Small_Cell_Architecture_Patrice_Nivaggioli_Consulting_System_Engineer_SP_EMEAR.pdf. [Accessed: Jul. 13, 2019]
- [11] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *Proceedings of the MCC workshop on Mobile Cloud Computing*, New York, NY, 2012, pp. 13–16.
- [12] H. H. Hoos and T. Stützle, “Scheduling Problems,” in *Stochastic Local Search: Foundations and Applications*, San Francisco: Morgan Kaufmann, 2005, pp.

417–465.

- [13] R. Aburukba, H. Ghenniwa, and W. Shen, “Agent-based approach for dynamic scheduling in content-based networks,” in *2006 IEEE International Conference on e-Business Engineering*, Shanghai, 2006, pp. 425–432.
- [14] B. L. Pik Lik, *Interdependence Between Agents in Multi Agent Systems*. M.S thesis. Western Australia: Curtin University, 2014. [Online]. Available: Curtin's Institutional Repository Espace. [Accessed: Jul. 10, 2019].
- [15] Sohal, A. S., Sandhu, R., Sood, S. K., & Chang, V. "A Cybersecurity Framework to Identify Malicious Edge Device in Fog Computing and Cloud-Of-Things Environments," *Computers & Security*, vol. 74, pp. 340–354, 2018.
- [16] G. Fortino and P. Trunfio, *Internet of Things Based on Smart Objects*, Switzerland: Springer, 2014.
- [17] J. Oueis, E. C. Strinati and S. Barbarossa, “The fog balancing: load distribution for small cell cloud computing,” *2015 IEEE 81st Vehicular Technology Conference*, Glasgow, 2015, pp. 1-6.
- [18] The Things Industries, “LoRaWAN | The Things Network,” *thethingsnetwork.org*, [Online]. Available: <https://www.thethingsnetwork.org/docs/lorawan>. [Accessed: May 10, 2019].
- [19] N. Gupta, “5G Technology and Requirement,” Mar. 30, 2019. [Online]. Available: <https://medium.com/swlh/5g-technology-and-requirement-d163434a5c45>. [Accessed: May 10, 2019].
- [20] G. Lo Nigro, S. Noto La Diega, G. Perrone, and P. Renna, “Coordination Policies to Support Decision Making in Distributed Production Planning,” *Robotics and Computer-Integrated Manufacturing*, vol. 19, no. 6, pp. 521–531, December 2003.
- [21] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal, and Y. Robert, “Centralized Versus Distributed Schedulers for Bag-Of-Tasks Applications,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 5, pp. 698–709, 2008.
- [22] G. Jules and M. Saadat, “Agent Cooperation Mechanism for Decentralized Manufacturing Scheduling,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 12, pp. 3351–3362, 2016.
- [23] E. O. Oyetunji, “Some Common Performance Measures in Scheduling Problems: Review Article,” *Research Journal of Applied Sciences, Engineering and Technology*, vol. 1, no. 2, pp. 6–9, 2009.
- [24] A. Mas-Colell, J. R. Green, C. Hara, I. Segal, S. Tadelis, and M. D. Whinston, *Microeconomic theory*. Oxfordshire: Oxford University Press, 1995.
- [25] D. Fudenberg and J. Tirole, *Game Theory*. Cambridge: MIT Press, 1991.
- [26] J. Nash, “Non-Cooperative Games,” *Annals of Mathematics*, vol. 54, no. 2, pp. 286-295, 1951.

- [27] I. Brocas, J. D. Carrillo and A. Sachdeva, “The path to equilibrium in sequential and simultaneous games: A mousetracking study,” *Journal of Economic Theory*, vol. 178, pp. 246-274, 2018.
- [28] P. C. Cramton, Y. Shoham, and R. Steinberg, *Combinatorial auctions*. Cambridge: MIT Press, 2006.
- [29] S. Li, Q. Ni, Y. Sun, G. Min, and S. Al-Rubaye, “Energy-Efficient Resource Allocation for Industrial Cyber-Physical IoT Systems in 5G Era,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 6, pp. 2618–2628, 2018.
- [30] H. Mei, K. Wang, and K. Yang, “Multi-Layer Cloud-RAN with Cooperative Resource Allocations for Low-Latency Computing and Communication Services,” *IEEE Access*, vol. 5, pp. 19023–19032, 2017.
- [31] T. C. Chiu, W. H. Chung, A. C. Pang, Y. J. Yu, and P. H. Yen, “Ultra-low latency service provision in 5G Fog-Radio Access Networks,” in *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications*, Valencia, 2016, pp. 1-6.
- [32] “What are C-RAN small cells? | CommScope.” [Online]. Available: <https://www.commscope.com/solutions/what-are-c-ran-small-cells/>. [Accessed: Dec. 11, 2018].
- [33] Z. Tayq, *Fronthaul integration and monitoring in 5G networks*. Ph.D thesis. Limoges, France: University of Limoges, 2018. [Online]. Available: ResearchGate Database. [Accessed: Jan. 15, 2019].
- [34] S. Shew, “Transport network support of IMT-2020/5G,” *Telecommunication Standardization sector of ITU*, Oct. 19, 2018. [Online]. Available: https://www.itu.int/dms_pub/itu-t/opb/tut/T-TUT-HOME-2018-2-PDF-E.pdf. [Accessed: Sept. 1, 2019].
- [35] D. Chitimalla, K. Kondepu, L. Valcarengi, M. Tornatore and B. Mukherjee, “5G fronthaul-latency and jitter studies of CPRI over ethernet,” in *IEEE/OSA Journal of Optical Communications and Networking*, vol. 9, no. 2, pp. 172-182, February 2017.
- [36] Ericsson AB, Huawei Technologies Co. Ltd, NEC Corporation, Alcatel Lucent and Nokia Networks, “CPRI Specification V7.0,” Oct. 2015. [Online]. Available: http://www.cpri.info/downloads/CPRI_v_7_0_2015-10-09.pdf [Accessed: Sept. 2, 2019].
- [37] P. F. Dutot, G. Mounié, and D. Trystram, “Scheduling Parallel Tasks: Approximation Algorithms,” in *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, J. T. Leung, Ed. Boca Raton: CRC Press, 2004, pp. 1–39.
- [38] “ILOG CPLEX Optimization Studio - Overview - | IBM.” [Online]. Available: <https://www.ibm.com/ae-en/products/ilog-cplex-optimization-studio>. [Accessed: May 22, 2019].
- [39] S. Adhau, M. L. Mittal and A. Mittal, “A Multi-Agent System for Distributed Multi-Project Scheduling: An Auction-Based Negotiation Approach,”

Engineering Applications of Artificial Intelligence, vol. 25, no. 8, pp. 1738–1751, 2012.

- [40] Z. Alibhai, “Contract Net Protocol.” [Online]. Available: [http://www2.ensc.sfu.ca/research/irms/courses/files/Contract Net Protocol.pdf](http://www2.ensc.sfu.ca/research/irms/courses/files/Contract_Net_Protocol.pdf). [Accessed: Dec. 11, 2018].
- [41] D. Bertsimas and J. Tsitsiklis, “Simulated Annealing,” *Statistical Science*, vol. 8, no. 1. pp. 10–15, 1993.
- [42] B. Farahani, F. Firouzi, V. Chang, M. Badaroglu, N. Constant and K. Mankodiya, “Towards Fog-Driven IoT eHealth: Promises and Challenges of IoT in Medicine and Healthcare,” *Future Generation Computer Systems*, vol. 78, pp. 659–676, 2018.
- [43] F. A. Kraemer, A. E. Braten, N. Tamkittikhun and D. Palma, “Fog Computing in Healthcare-A Review and Discussion,” *IEEE Access*, vol. 5, pp. 9206–9222, 2017.
- [44] O. Fratu, C. Pena, R. Craciunescu and S. Halunga, “Fog computing system for monitoring Mild Dementia and COPD patients - Romanian case study,” in *2015 12th International Conference on Telecommunication in Modern Satellite, Cable and Broadcasting Services*, Nis, 2015, pp. 123–128.
- [45] B. McCarthy, D. Casey, D. Devane, K. Murphy, E. Murphy and Y. Lacasse, “Pulmonary Rehabilitation For Chronic Obstructive Pulmonary Disease,” *Cochrane Database of Systematic Reviews*, vol. 2, pp. 1465-75, February 2015.
- [46] W. T. Liu, “Efficacy of A Cell Phone-Based Exercise Programme for COPD,” *The European respiratory journal*, vol. 32, pp. 651-9, June 2008.
- [47] X. Masip-Bruin, E. Marín-Tordera, A. Alonso and J. Garcia, “Fog-to-cloud Computing (F2C): The key technology enabler for dependable e-health services deployment,” in *2016 Mediterranean Ad Hoc Networking Workshop*, Vilanova i la Geltru, 2016, pp. 1-5.

Appendix A – Scheduling Problem Model Validation

The hand calculation for allocation example 1. Calculation for the objective function:

$$\begin{aligned}
 & t_{1,3}^{start} * t_{slot} + \frac{P_{1,3}}{\sum_i^I \sum_p^P x_{1,3,i,p,t_{1,3}^{start}} * \Delta P} \\
 & + t_{2,3}^{start} * t_{slot} + \frac{P_{2,3}}{\sum_i^I \sum_p^P x_{2,3,i,p,t_{2,3}^{start}} * \Delta P} \\
 & + t_{3,3}^{start} * t_{slot} + \frac{P_{3,3}}{\sum_i^I \sum_p^P x_{3,3,i,p,t_{3,3}^{start}} * \Delta P} \\
 & = 2 * 10^{-3} \\
 & + 2.6667 * 10^{-3} \\
 & + 2.6667 * 10^{-3} \\
 & \text{Sum of makespan} = 7.3334 \text{ ms} \\
 & \text{Avg. makespan} = \frac{7.3334 \text{ ms}}{3} = 2.4445 \text{ ms}
 \end{aligned}$$

Checking if all the constraints are respected starting with the constraint (10):

$$\begin{aligned}
 \sum_j^J \sum_k^{K_j} x_{jk ipt} &\leq 1, \quad \forall i, \forall p, \forall t \\
 \sum_{j=1}^3 \sum_{k=1}^3 x_{jk ipt} &\leq 1, \quad \forall i, \forall p, \forall t
 \end{aligned}$$

For $p = 1, t = 0$

For $i = 1$

$$\begin{aligned}
 & = x_{1,1,1,1,0} + x_{1,2,1,1,0} + x_{1,3,1,1,0} + x_{2,1,1,1,0} + x_{2,2,1,1,0} + x_{2,3,1,1,0} + x_{3,1,1,1,0} \\
 & \quad + x_{3,2,1,1,0} \\
 & = 1 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 \leq 1
 \end{aligned}$$

$$1 \leq 1$$

The constraint is satisfied for the first compute block in the first fog. Similarly, the same can be applied to $i = 1, 2$, $t = 0, 1, 2, 3$ and $p = 1, 2, 3, 4, 5$. The result is represented in Table A.1 below.

Table A.1: Allocations of each compute block to all operations

Indices			$\sum_{j=1}^3 \sum_{k=1}^3 x_{jk ipt}$
$i = 1$	$t = 0$	$p = 1$	1
		$p = 2$	1
		$p = 3$	1
		$p = 4$	1
		$p = 5$	1
	$t = 1$	$p = 1$	1
		$p = 2$	1
		$p = 3$	1
		$p = 4$	1
		$p = 5$	0
	$t = 2$	$p = 1$	1
		$p = 2$	1
		$p = 3$	1
		$p = 4$	1
		$p = 5$	1
$t = 3$	$p = 1$	1	
	$p = 2$	1	
	$p = 3$	1	
	$p = 4$	0	
	$p = 5$	0	
$i = 2$	$t = 0$	$p = 1$	1
		$p = 2$	1
		$p = 3$	1
		$p = 4$	1
		$p = 5$	0
	$t = 1$	$p = 1$	1
		$p = 2$	1
		$p = 3$	1
		$p = 4$	1
		$p = 5$	1
	$t = 2$	$p = 1$	1
		$p = 2$	1
		$p = 3$	1
		$p = 4$	1
		$p = 5$	1
$t = 3$	$p = 1$	1	
	$p = 2$	1	
	$p = 3$	0	
	$p = 4$	0	
	$p = 5$	0	

This confirms the constraint (10). The second constraint to check is (11) that confirms that each operation shall be executed in by no more than one fog only. The constraint is shown to be satisfied given the allocation example 1 as follows:

$$\begin{aligned}
& \sum_i^I \mathfrak{S}_i(O_{jk}) \leq 1, \quad \forall j, \forall k \\
& = \sum_{i=1}^2 \mathfrak{S}_i(O_{jk}) \leq 1, \quad \forall j, \forall k \\
& = \mathfrak{S}_1(O_{jk}) + \mathfrak{S}_2(O_{jk}) \leq 1, \quad \forall j, \forall k
\end{aligned}$$

For $j = 1, k = 1$

$$\mathfrak{S}_1(O_{1,1}) = \begin{cases} 1, & \sum_{p=1}^5 \sum_{t=0}^3 x_{jk ipt} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Since

$$\begin{aligned}
\sum_{p=1}^5 \sum_{t=0}^3 x_{jk ipt} &= x_{1,1,1,1,0} + x_{1,1,1,1,1} + x_{1,1,1,1,2} + x_{1,1,1,1,3} + x_{1,1,1,2,0} + x_{1,1,1,2,1} \\
&+ x_{1,1,1,2,2} + x_{1,1,1,2,3} + x_{1,1,1,3,0} + x_{1,1,1,3,1} + x_{1,1,1,3,2} + x_{1,1,1,3,3} \\
&+ x_{1,1,1,4,0} + x_{1,1,1,4,1} + x_{1,1,1,4,2} + x_{1,1,1,4,3} + x_{1,1,1,5,0} + x_{1,1,1,5,1} \\
&+ x_{1,1,1,5,2} + x_{1,1,1,5,3} \\
&= 1 + 0 + 0 + 0 + 1 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 \\
&+ 0 + 0 + 0 + 0 = 2
\end{aligned}$$

then

$$\mathfrak{S}_1(O_{1,1}) = \begin{cases} 1, & 2 > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\mathfrak{S}_1(O_{1,1}) = 1$$

And for the second fog

$$\mathfrak{F}_2(O_{11}) = \begin{cases} 1, & \sum_{p=1}^5 \sum_{t=0}^3 x_{jk ipt} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Since

$$\begin{aligned} \sum_{p=1}^5 \sum_{t=0}^3 x_{jk ipt} &= x_{1,1,2,1,0} + x_{1,1,2,1,1} + x_{1,1,2,1,2} + x_{1,1,2,1,3} + x_{1,1,2,2,0} + x_{1,1,2,2,1} \\ &+ x_{1,1,2,2,2} + x_{1,1,2,2,3} + x_{1,1,2,3,0} + x_{1,1,2,3,1} + x_{1,1,2,3,2} + x_{1,1,2,3,3} \\ &+ x_{1,1,2,4,0} + x_{1,1,2,4,1} + x_{1,1,2,4,2} + x_{1,1,2,4,3} + x_{1,1,2,5,0} + x_{1,1,2,5,1} \\ &+ x_{1,1,2,5,2} + x_{1,1,2,5,3} \\ &= 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 \\ &+ 0 + 0 + 0 + 0 = 0 \end{aligned}$$

Then

$$\mathfrak{F}_2(O_{1,1}) = \begin{cases} 1, & 0 > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\mathfrak{F}_2(O_{1,1}) = 0$$

Adding the value of two small cells

$$\mathfrak{F}_1(O_{jk}) + \mathfrak{F}_2(O_{jk}) \leq 1$$

$$1 + 0 \leq 1$$

And this satisfies the constraint but only for the operation $O_{1,1}$. Similarly, the constraint can be checked for the other operations to get the values in Table A.2.

Table A.2: Allocations of each fog to all operations

Indices		$\sum_{p=1}^5 \sum_{t=0}^3 x_{jk ipt}$ For $i = 1$	$\sum_{p=1}^5 \sum_{t=0}^3 x_{jk ipt}$ For $i = 2$	$\mathfrak{F}_1(O_{jk})$	$\mathfrak{F}_2(O_{jk})$
$j = 1$	$k = 1$	2	0	1	0
	$k = 2$	4	0	1	0
	$k = 3$	5	0	1	0
$j = 2$	$k = 1$	3	0	1	0
	$k = 2$	0	2	0	1
	$k = 3$	0	4	0	1
$j = 3$	$k = 1$	0	4	0	1
	$k = 2$	0	6	0	1
	$k = 3$	3	0	1	0

From the values acquired in Table A.2, it is noticeable that only one of the small cells will return a value of 1 which satisfies the constraint (11). Furthermore, the dependency constraint is checked (12). No need to check the inequality for $x_{ijkpt} = 0$ since the right-hand side is either 0 or 1. Thus, $x_{ijkpt} = 0$ will never violate the constraint. In addition, for operations O_{jk} where $k = 1$ there is no dependency since it is the first operations in the request j . So, the check process shall start with operations $k = 2,3$ of requests $j = 1,2,3$ and for all $x_{ijkpt} = 1$. The resultant values are represented in Table A.3.

Table A.3: Dependency constraint check

Allocations	h	γ_{jkh}	E_{jht}	$ (\gamma_{jkh} * E_{jht} - 1) - 1 $
$x_{i,1,2,p,1}$	1	1	1	1
	3	0	0	1
$x_{i,1,3,p,2}$	1	0	1	1
	2	1	1	1
$x_{i,2,2,p,1}$	1	1	1	1
	3	0	0	1
$x_{i,2,3,p,2}$	1	0	1	1
	2	1	1	1
$x_{i,3,2,p,1}$	1	1	1	1
	3	0	0	1
$x_{i,3,3,p,3}$	1	0	1	1
	2	1	1	1

From Table A.3, all allocations satisfied the dependency constraint (12). Thus, dependency constraint is check. Next, the deadline constraint is to be checked. The second term $(D_T(j) + D_p)$ is a given number to the model so this actually can be removed as follows: let $\tau'_j = \tau_j - D_T(j) + D_p$. Accordingly, constraint (13) can be represented as follows:

$$C_j < \tau'_j, \forall j$$

For the sake of this example the transmission delay is considered to be zero such that $\tau'_j = \tau_j, \forall j$. The term \max_k indicates the last operation that is executed for each request. Thus, the last operation for every request $(O_{1,3}, O_{2,3}, O_{3,3})$ must be checked if they are executed before the deadline. For $O_{1,3}$:

$$\begin{aligned}
& t_{1,3}^{start} * t_{duration} + \frac{P_{1,3}}{\sum_i^I \sum_p^P x_{1,3,i,p,t_{1,3}^{start}} * \Delta P} \\
&= (2) * (666.7 * 10^{-6}) + \frac{5 * 10^5}{\sum_i^I \sum_p^P x_{1,3,i,p,t_{1,3}^{start}} * (150 * 10^6)} \\
&= (2) * (666.7 * 10^{-6}) \\
&+ \frac{5 * 10^5}{(x_{1,3,1,1,2} + x_{1,3,1,2,2} + x_{1,3,1,3,2} + x_{1,3,1,4,2} + x_{1,3,1,5,2} + \dots + 0) * (150 * 10^6)} \\
&= (2) * (666.7 * 10^{-6}) + \frac{5 * 10^5}{(0 + \dots + 1 + 1 + 1 + 1 + 1 + \dots + 0) * (150 * 10^6)} \\
&= (2) * (666.7 * 10^{-6}) + \frac{5 * 10^5}{(5) * (150 * 10^6)} \\
&= 2 \text{ ms}
\end{aligned}$$

$$\tau_1 = 4 \text{ time slots}(0 \rightarrow 3) = 4 * (666.7 \mu s) = 2.6668 \text{ ms}$$

$$2 \text{ ms} \leq 2.6668 \text{ ms}$$

For $O_{2,3}$

$$\begin{aligned}
& t_{2,3}^{start} * t_{slot} + \frac{P_{2,3}}{\sum_i^I \sum_p^P x_{2,3,i,p,t_{2,3}^{start}} * \Delta P} \\
&= (2) * (666.7 * 10^{-6}) + \frac{4 * 10^5}{\sum_i^I \sum_p^P x_{2,3,i,p,t_{2,3}^{start}} * (150 * 10^6)} \\
&= (2) * (666.7 * 10^{-6}) + \frac{4 * 10^5}{(0 + \dots x_{2,3,2,1,2} + x_{2,3,2,2,2} + \dots 0) * (150 * 10^6)} \\
&= (2) * (666.7 * 10^{-6}) = + \frac{4 * 10^5}{(0 + \dots + 1 + 1 + \dots + 0) * (150 * 10^6)} \\
&= (2) * (666.7 * 10^{-6}) + \frac{4 * 10^5}{(2) * (150 * 10^6)} \\
&= 2.6667 \text{ ms}
\end{aligned}$$

$$\tau_2 = 4 \text{ time slots}(0 \rightarrow 3) = 4 * (666.7 \mu s) = 2.6668 \text{ ms}$$

$$2.6667 \text{ ms} \leq 2.6668 \text{ ms}$$

And for $O_{3,3}$

$$\begin{aligned}
& t_{3,3}^{start} * t_{slot} + \frac{P_{3,3}}{\sum_i^I \sum_p^P x_{3,3,i,p,t_{3,3}^{start}} * \Delta P} \\
&= (3) * (666.7 * 10^{-6}) + \frac{3 * 10^5}{\sum_i^I \sum_p^P x_{3,3,i,p,t_{3,3}^{start}} * (150 * 10^6)} \\
&= (3) * (666.7 * 10^{-6}) + \frac{3 * 10^5}{(x_{3,3,1,1,3} + x_{3,3,1,2,3} + x_{3,3,1,3,3} + \dots + 0) * (150 * 10^6)} \\
&= (3) * (666.7 * 10^{-6}) + \frac{3 * 10^5}{(1 + 1 + 1 + \dots + 0) * (150 * 10^6)} \\
&= (3) * (666.7 * 10^{-6}) + \frac{3 * 10^5}{(3) * (150 * 10^6)} \\
&= 2.6667 \text{ ms}
\end{aligned}$$

$$\tau_3 = 4 \text{ time slots}(0 \rightarrow 3) = 4 * (666.7 \mu\text{s}) = 2.6668 \text{ ms}$$

$$2.6667 \text{ ms} \leq 2.6668 \text{ ms}$$

All request will be finished before the deadline, so the constraint is satisfied. Finally, constraint (14) should be checked which guarantees that any compute block allocated for any operation shall not be freed until all its instructions are executed, i.e. no preemption. There are two test cases for this constraint. First, the operations that are executed during one time slots. Secondly, the operations that are executed during multiple timeslots. Let us consider $O_{1,3}$ for the first case:

$$|x_{jkip,t} - x_{jkip,t+1}| = |E_{jk,t} - E_{jk,t+1}|, \quad t_{jk}^{start} \leq t < T, \forall j, \forall k, \forall i, \forall p$$

$$|x_{1,3,1,1,2} - x_{1,3,1,1,3}| = |E_{1,3,2} - E_{1,3,3}|$$

$$|1 - 0| = |0 - 1|$$

$$1 = 1$$

This satisfies the constraint for the operation $O_{1,3}$. For the second case, consider $O_{3,2}$ which is allocated during $t = 1,2$. For $p = 3, t = 1$:

$$|x_{3,2,2,2,1} - x_{3,2,2,2,2}| = |E_{3,2,1} - E_{3,2,2}|$$

$$|1 - 1| = |0 - 0|$$

$$0 = 0$$

For $p = 3, t = 2$

$$|x_{3,2,2,2,2} - x_{3,2,2,2,3}| = |E_{3,2,2} - E_{3,2,3}|$$

$$|1 - 0| = |0 - 1|$$

$$1 = 1$$

The same can be applied for all operations and their corresponding compute blocks. We can conclude that this constraint is also satisfied.

The workout of allocation example 2:

Calculation for the objective function:

$$\begin{aligned}
& t_{1,3}^{start} * t_{slot} + \frac{P_{1,3}}{\sum_i^I \sum_p^P x_{1,3,i,p,t_{1,3}^{start}} * \Delta P} \\
& + t_{2,3}^{start} * t_{slot} + \frac{P_{2,3}}{\sum_i^I \sum_p^P x_{2,3,i,p,t_{2,3}^{start}} * \Delta P} \\
& + t_{3,3}^{start} * t_{slot} + \frac{P_{3,3}}{\sum_i^I \sum_p^P x_{3,3,i,p,t_{3,3}^{start}} * \Delta P} \\
& = 3 * (666.7 * 10^{-6}) + \frac{5 * 10^5}{\sum_i^I \sum_p^P x_{1,3,i,p,t_{1,3}^{start}} * (150 * 10^6)} \\
& + 2 * (666.7 * 10^{-6}) + \frac{4 * 10^5}{\sum_i^I \sum_p^P x_{2,3,i,p,t_{1,3}^{start}} * (150 * 10^6)} \\
& + 2 * (666.7 * 10^{-6}) + \frac{3 * 10^5}{\sum_i^I \sum_p^P x_{3,3,i,p,t_{3,3}^{start}} * (150 * 10^6)} \\
& = 3 * (666.7 * 10^{-6}) + \frac{5 * 10^5}{5 * (150 * 10^6)}
\end{aligned}$$

$$+2 * (666.7 * 10^{-6}) + \frac{4 * 10^5}{4 * (150 * 10^6)}$$

$$+2 * (666.7 * 10^{-6}) + \frac{3 * 10^5}{3 * (150 * 10^6)}$$

$$= 2.6667 * 10^{-3}$$

$$+2 * 10^{-3}$$

$$+2 * 10^{-3}$$

Sum of makespan = 6.6667 ms

$$\text{Avg. makespan} = \frac{6.6667 \text{ ms}}{3} = 2.222 \text{ ms}$$

Appendix B – Self-Classification and Local Scheduling Model Validation

The calculation for example 1 used in the validation process of the self-classification model:

First, the term C_j as follows is evaluated:

$$C_j = \max_k \left(t_{jk}^s * t_d + \frac{P_{jk}}{\sum_p x_{jkp, t_{jk}^s} * \Delta P} \right)$$

The term \max_k indicates the last operation. Thus, we can apply our calculation for $(O_{1,4}, O_{2,4})$ as the following:

$$C_1 = t_{1,4}^s * t_d + \frac{P_{1,4}}{\sum_p x_{1,4,p, t_{1,4}^s} * \Delta P}$$

$$C_1 = 4 * (666.7 * 10^{-6}) + \frac{6 * 10^5}{2 * (150 * 10^6)}$$

$$C_1 = 4.6667 * 10^{-3}$$

$$C_2 = t_{2,4}^s * t_d + \frac{P_{2,4}}{\sum_p x_{2,4,p, t_{2,4}^s} * \Delta P}$$

$$C_2 = 6 * (666.7 * 10^{-6}) + \frac{3 * 10^5}{3 * (150 * 10^6)}$$

$$C_2 = 4.6667 * 10^{-3}$$

Then the first mathematical summation term of the objective function is evaluated as follows:

$$\sum_j^J |\tau_j - C_j| = |\tau_1 - C_1| + |\tau_2 - C_2|$$

$$= |(6 + 1) * 666.7 * 10^{-6} - 4.6667 * 10^{-3}| + |6 * 666.7 * 10^{-6} - 4.6667 * 10^{-3}|$$

$$= |(6 + 1) * 666.7 * 10^{-6} - 4.6667 * 10^{-3}| + |6 * 666.7 * 10^{-6} - 4.6667 * 10^{-3}|$$

$$= |(6 + 1) * 666.7 * 10^{-6} - 4.6667 * 10^{-3}| + |6 * 666.7 * 10^{-6} - 4.6667 * 10^{-3}|$$

$$\sum_j^J |\tau_j - C_j| = 0$$

Lastly, the second summation term in the objective function is calculated as follows:

$$d_1 = \begin{cases} 1, & C_1 > \tau_1 \\ 0, & \text{otherwise} \end{cases}$$

$$d_1 = \begin{cases} 1, & 4.6667 * 10^{-3} > 4.6667 * 10^{-3} \\ 0, & \text{otherwise} \end{cases}$$

$$d_1 = 0$$

Likewise,

$$d_2 = \begin{cases} 1, & C_2 > \tau_2 \\ 0, & \text{otherwise} \end{cases}$$

$$d_2 = \begin{cases} 1, & 4.6667 * 10^{-3} > 4.6667 * 10^{-3} \\ 0, & \text{otherwise} \end{cases}$$

$$d_2 = 0$$

Thus,

$$\sum_j^J (d_j + 1) = 2$$

$$\sum_j^J |\tau_j - C_j| * \sum_j^J (d_j + 1) = 0 * 2 = 0$$

The calculation for example 2 used in the validation process of the self-classification model:

For simplicity, it is noticeable from the previous example that the objective value is “0” for any request finishes at its deadline. Therefore, we can resolve $C_1 = C_2 = C_4 = C_5 = 0$. Consequently:

$$\sum_j^J |\tau_j - C_j| * \sum_j^J (d_j + 1) = |\tau_3 - C_3| * (d_3 + 1)$$

$$\begin{aligned}
&= \left| (9 + 1) * 666.7 * 10^{-6} - 11 * (666.7 * 10^{-6}) + \frac{3 * 10^5}{3 * (150 * 10^6)} \right| * (1 + 1) \\
&= |2 * 666.7 * 10^{-6}| * (2) \\
&= 4 \text{ time slots}
\end{aligned}$$

The calculation for example 1 used in the validation process of the local scheduling model:

$$\begin{aligned}
C_j &= \max_k \left(t_{jk}^s * t_d + \frac{P_{jk}}{\sum_p x_{jkp, t_{jk}^s} * \Delta P} \right) \\
\sum_j C_j &= C_1 + C_2
\end{aligned}$$

Solving for C_1 ,

$$\begin{aligned}
C_1 &= t_{1,4}^s * t_d + \frac{P_{1,4}}{\sum_p x_{1,4,p, t_{1,4}^s} * \Delta P} \\
C_1 &= 4 * 666.7 * 10^{-6} + \frac{5 * 10^5}{5 * 150 * 10^6} \\
C_1 &= 3.3334 \text{ ms} = 5 \text{ time slots}
\end{aligned}$$

Solving for C_2 ,

$$\begin{aligned}
C_2 &= t_{2,4}^s * t_d + \frac{P_{2,4}}{\sum_p x_{2,4,p, t_{2,4}^s} * \Delta P} \\
C_2 &= 1 * 666.7 * 10^{-6} + \frac{3 * 10^5}{3 * 150 * 10^6} \\
C_2 &= 1.3334 \text{ ms} = 2 \text{ time slots}
\end{aligned}$$

Thus,

$$\sum_j C_j = C_1 + C_2 = 3.3334 + 1.3334 = 4.6668 \text{ ms} = 7 (666.7 \text{ ms}) = 7 \text{ time slots}$$

The calculation for example 1 used in the validation process of the winner determination model:

$$U_j(a) = \sum_k^{K_j} \frac{P_{jk}}{\Delta P} + t^{max} - C_j^{min}(a)$$

$$U_1(a) = \sum_k^{K_1} \frac{P_{1,k}}{\Delta P} + t^{max} - t_{1,4}^s - \text{ceil} \left(\frac{P_{jk}}{\sum_p^P x_{1,4,1,p,t_{1,4}^s} * \Delta P * t_d} \right)$$

$$U_1(a) = 12 + 9 - 7 - \text{ceil} \left(\frac{(3 * 10^5)}{(3) * (150 * 10^6) * (666.7 * 10^{-6})} \right)$$

$$U_1(a) = 12 + 9 - 7 - 1 = 13 \text{ currency units}$$

The calculation for example 2 used in the validation process of the winner determination model:

$$U_j(a) = \sum_k^{K_j} \frac{P_{jk}}{\Delta P} + t^{max} - C_j^{min}(a)$$

$$U_1(a) = \sum_k^{K_1} \frac{P_{1k}}{\Delta P} + t^{max} - t_{1,4}^s - \text{ceil} \left(\frac{P_{jk}}{\sum_p^P x_{1,4,a,p,t_{1,4}^s} * \Delta P * t_d} \right)$$

$$U_1(a) = 12 + 9 - 5 - \text{ceil} \left(\frac{(3 * 10^5)}{(3) * (150 * 10^6) * (666.7 * 10^{-6})} \right)$$

$$U_1(a) = 12 + 9 - 5 - 1 = 15 \text{ currency units}$$

Appendix C – Winner Determination Model Validation

The utility calculation for selling the good {3,5,6} to the bid {2,5} and {2,4}:

$$EET(\{2,5\}, \{3,5,6\}) = \max\left(\text{ceil}\left(\frac{\text{req. blocks}(\{2,5\})}{\text{num. of CBs}(\{3,5,6\})}\right), \text{dependencyLvl}(\{2,5\})\right)$$

$$EET(\{2,5\}, \{3,5,6\}) = \max\left(\text{ceil}\left(\frac{2 + 3 + 4}{6}\right), 2\right)$$

$$EET(\{2,5\}, \{3,5,6\}) = 2$$

Then,

$$EFT(\{2,5\}, \{3,5,6\}) = EET(\{2,5\}, \{3,5,6\}) + \text{free starting time}(\{3,5,6\}) - 1$$

$$EFT(\{2,5\}, \{3,5,6\}) = 2 + 5 - 1 = 6$$

where EET and EFT are the estimated execution time and the estimated finishing time, respectively. Since the estimated finishing time is not passing the deadline, this deal shall be considered. Therefore,

$$\begin{aligned} \text{Utility}(\{2,5\}, \{3,5,6\}) &= \frac{\text{req. blocks}(\{2,5\})}{\text{num. of CBs}(\{3,5,6\})} + 2 * \text{time window} \\ &\quad - EFT(\{2,5\}, \{3,5,6\}) - \text{deadline}(\{2,5\}) \end{aligned}$$

$$\text{Utility}(\{2,5\}, \{3,5,6\}) = \frac{2 + 3 + 4}{6} + 2 * 8 - 6 - 6 = 5$$

Similarly, the utility for the buyer request {2,4} can be obtained as follows:

$$EET(\{2,4\}, \{3,5,6\}) = \max\left(\text{ceil}\left(\frac{\text{req. blocks}(\{2,4\})}{\text{num. of CBs}(\{3,5,6\})}\right), \text{dependencyLvl}(\{2,4\})\right)$$

$$EET(\{2,4\}, \{3,5,6\}) = \max\left(\text{ceil}\left(\frac{4 + 2 + 4 + 5}{6}\right), 2\right)$$

$$EET(\{2,4\}, \{3,5,6\}) = 3$$

Then,

$$EFT(\{2,4\}, \{3,5,6\}) = EET(\{2,4\}, \{3,5,6\}) + \text{free starting time}(\{3,5,6\}) - 1$$

$$EFT(\{2,4\}, \{3,5,6\}) = 3 + 5 - 1 = 7$$

Since the estimated finishing time is not passing the deadline, this deal shall also be considered. Consequently,

$$\begin{aligned} \text{Utility}(\{2,4\}, \{3,5,6\}) &= \frac{\text{req. blocks}(\{2,4\})}{\text{num. of CBs}(\{3,5,6\})} + 2 * \text{time window} \\ &\quad - \text{EFT}(\{2,4\}, \{3,5,6\}) - \text{deadline}(\{2,4\}) \\ \text{Utility}(\{2,4\}, \{3,5,6\}) &= \frac{4 + 2 + 4 + 5}{6} + 2 * 8 - 7 - 8 = 3 \end{aligned}$$

The utility calculation for selling the good {1,3,6} to the bid {2,4}:

$$\begin{aligned} \text{EET}(\{2,4\}, \{1,3,6\}) &= \max\left(\text{ceil}\left(\frac{\text{req. blocks}(\{2,4\})}{\text{num. of CBs}(\{1,3,6\})}\right), \text{dependencyLvl}(\{2,4\})\right) \\ \text{EET}(\{2,4\}, \{1,3,6\}) &= \max\left(\text{ceil}\left(\frac{4 + 2 + 4 + 5}{6}\right), 2\right) \\ \text{EET}(\{2,4\}, \{1,3,6\}) &= 3 \end{aligned}$$

Then,

$$\begin{aligned} \text{EFT}(\{2,4\}, \{1,3,6\}) &= \text{EET}(\{2,4\}, \{1,3,6\}) + \text{free starting time}(\{1,3,6\}) - 1 \\ \text{EFT}(\{2,4\}, \{1,3,6\}) &= 3 + 3 - 1 = 5 \end{aligned}$$

Since the estimated finishing time is not passing the deadline, this deal shall also be considered. Therefore,

$$\begin{aligned} \text{Utility}(\{2,4\}, \{1,3,6\}) &= \frac{\text{req. blocks}(\{2,4\})}{\text{num. of CBs}(\{1,3,6\})} + 2 * \text{time window} \\ &\quad - \text{EFT}(\{2,4\}, \{1,3,6\}) - \text{deadline}(\{2,4\}) \\ \text{Utility}(\{2,4\}, \{1,3,6\}) &= \frac{4 + 2 + 4 + 5}{6} + 2 * 8 - 5 - 8 = 5 \end{aligned}$$

Vita

Ahmed Fahmy was born in 1995, in Abu Dhabi, United Arab Emirates. He received his primary and secondary education in Abu Dhabi, UAE. He received his B.Sc. degree in Computer Engineering from the University of Sharjah in 2017. In 2018, he interned as a system engineer in DELL, RSA Security.

In January 2018, he joined the Computer Engineering master's program in the American University of Sharjah as a graduate teaching assistant. During his master's study, he co-authored a few papers which were presented in international conferences. His research interests are in cloud computing, fog computing, 5G network data structures, optimization, embedded systems and microcontrollers, IoT.