Real-Time Resource Optimization for Wireless Networks

Yan Huang

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Electrical Engineering

Y. Thomas Hou, Chair
Wenjing Lou
Jeffrey H. Reed
Wu-chun Feng
Weijun Xie

December 16, 2020
Blacksburg, Virginia

Keywords: wireless network, resource allocation, scheduling, mathematical modeling, optimization, real time, GPU, deep learning, deep reinforcement learning

Real-Time Resource Optimization for Wireless Networks

Yan Huang

(ABSTRACT)

Resource allocation in modern wireless networks is constrained by increasingly stringent real-time requirements. Such real-time requirements typically come from, among others, the short coherence time on a wireless channel, the small time resolution for resource allocation in OFDM-based radio frame structure, or the low-latency requirements from delay-sensitive applications. An optimal resource allocation solution is useful only if it can be determined and applied to the network entities within its expected time. For today's wireless networks such as 5G NR, such expected time (or real-time requirement) can be as low as 1 ms or even 100 μs.

Most of the existing resource optimization solutions to wireless networks do not explicitly take real-time requirement as a constraint when developing solutions. In fact, the mainstream of research works relies on the asymptotic complexity analysis for designing solution algorithms. Asymptotic complexity analysis is only concerned with the growth of its computational complexity as the input size increases (as in the big-O notation). It cannot capture the real-time requirement that is measured in wall-clock time. As a result, existing approaches such as exact or approximate optimization techniques from operations research are usually not useful in wireless networks in the field. Similarly, many problem-specific heuristic solutions with polynomial-time asymptotic complexities may suffer from a similar fate, if their complexities are not tested in actual wall-clock time.

To address the limitations of existing approaches, this dissertation presents novel real-time solution designs to two types of optimization problems in wireless networks: i) problems that have closed-form mathematical models, and ii) problems that cannot be modeled in closed-form. For the first type of problems, we propose a novel approach that consists

of (i) problem decomposition, which breaks an original optimization problem into a large number of small and independent sub-problems, (ii) search intensification, which identifies the most promising problem sub-space and selects a small set of sub-problems to match the available GPU processing cores, and (iii) GPU-based large-scale parallel processing, which solves the selected sub-problems in parallel and finds a near-optimal solution to the original problem. The efficacy of this approach has been illustrated by our solutions to the following two problems.

- **Real-Time Scheduling to Achieve Fair LTE/Wi-Fi Coexistence:** We investigate a resource optimization problem for the fair coexistence between LTE and Wi-Fi in the unlicensed spectrum. The real-time requirement for finding the optimal channel division and LTE resource allocation solution is on 1 ms time scale. This problem involves the optimal division of transmission time for LTE and Wi-Fi across multiple unlicensed bands, and the resource allocation among LTE users within the LTE's "ON" periods. We formulate this optimization problem as a mixed-integer linear program and prove its NP-hardness. Then by exploiting the unique problem structure, we propose a real-time solution design that is based on problem decomposition and GPU-based parallel processing techniques. Results from an implementation on the NVIDIA GPU/CUDA platform demonstrate that the proposed solution can achieve near-optimal objective and meet the 1 ms timing requirement in 4G LTE.

- **An Ultrafast GPU-based Proportional Fair Scheduler for 5G NR:** We study the popular proportional-fair (PF) scheduling problem in a 5G NR environment. The real-time requirement for determining the optimal (with respect to the PF objective) resource allocation and MCS selection solution is 125 μs (under 5G numerology 3). In this problem, we need to allocate frequency-time resource blocks on an operating channel and assign modulation and coding scheme (MCS) for each active user in the

cell. We present GPF+ — a GPU based real-time PF scheduler. With GPF+, the original PF optimization problem is decomposed into a large number of small and independent sub-problems. We then employ a cross-entropy based search intensification technique to identify the most promising problem sub-space and select a small set of sub-problems to fit into a GPU. After solving the selected sub-problems in parallel using GPU cores, we find the best sub-problem solution and use it as the final scheduling solution. Evaluation results show that GPF+ is able to provide near-optimal PF performance in a 5G cell while meeting the 125 μs real-time requirement.

For the second type of problems, where there is no closed-form mathematical formulation, we propose to employ model-free deep learning (DL) or deep reinforcement learning (DRL) techniques along with judicious consideration of timing requirement throughout the design. Under DL/DRL, we employ deep function approximators (neural networks) to learn the unknown objective function of an optimization problem, approximate an optimal algorithm to find resource allocation solutions, or discover important mapping functions related to the resource optimization. To meet the real-time requirement, we propose to augment DL or DRL methods with optimization techniques at the input or output of the deep function approximators to reduce their complexities and computational time. Under this approach, we study the following two problems:

- **A DRL-based Approach to Dynamic eMBB/URLLC Multiplexing in 5G NR:** We study the problem of dynamic multiplexing of eMBB and URLLC on the same channel through preemptive resource puncturing. The real-time requirement for determining the optimal URLLC puncturing solution is 1 ms (under 5G numerology 0). A major challenge in solving this problem is that it cannot be modeled using closed-form mathematical expressions. To address this issue, we develop a model-free DRL approach which employs a deep neural network to learn an optimal algorithm

to allocate the URLLC puncturing over the operating channel, with the objective of minimizing the adverse impact from URLLC traffic on eMBB. Our contributions include a novel learning method that exploits the intrinsic properties of the URLLC puncturing optimization problem to achieve a fast and stable learning convergence, and a mechanism to ensure feasibility of the deep neural network's output puncturing solution. Experimental results demonstrate that our DRL-based solution significantly outperforms state-of-the-art algorithms proposed in the literature and meets the 1 ms real-time requirement for dynamic multiplexing.

- **A DL-based Link Adaptation for eMBB/URLLC Multiplexing in 5G NR:** We investigate MCS selection for eMBB traffic under the impact of URLLC preemptive puncturing. The real-time requirement for determining the optimal MCSs for all eMBB transmissions scheduled in a transmission interval is 125 μs (under 5G numerology 3). The objective is to have eMBB meet a given block-error rate (BLER) target under the adverse impact of URLLC puncturing. Since this problem cannot be mathematically modeled in closed-form, we proposed a DL-based solution design that uses a deep neural network to learn and predict the BLERs of a transmission under each MCS level. Then based on the BLER predictions, an optimal MCS can be found for each transmission that can achieve the BLER target. To meet the 5G real-time requirement, we implement this design through a hybrid CPU and GPU architecture to minimize the execution time. Extensive experimental results show that our design can select optimal MCS under the impact of preemptive puncturing and meet the 125 μs timing requirement.

Real-Time Resource Optimization for Wireless Networks

Yan Huang

(GENERAL AUDIENCE ABSTRACT)

In modern wireless networks such as 4G LTE and 5G NR, the optimal allocation of radio resources must be performed within a real-time requirement of 1 ms or even 100 μs time scale. Such a real-time requirement comes from the physical properties of wireless channels, the short time resolution for resource allocation defined in the wireless communication standards, and the low-latency requirement from delay-sensitive applications.

Real-time requirement, although necessary for wireless networks in the field, has hardly been considered as a key constraint for solution design in the research community. Existing solutions in the literature mostly consider theoretical computational complexities, rather than actual computation time as measured by wall clock.

To address the limitations of existing approaches, this dissertation presents real-time solution designs to two types of optimization problems in wireless networks: i) problems that have mathematical models, and ii) problems that cannot be modeled mathematically. For the first type of problems, we propose a novel approach that consists of (i) problem decomposition, (ii) search intensification, and (iii) GPU-based large-scale parallel processing techniques. The efficacy of this approach has been illustrated by our solutions to the following two problems.

- **Real-Time Scheduling to Achieve Fair LTE/Wi-Fi Coexistence:** We investigate a resource optimization problem for the fair coexistence between LTE and Wi-Fi users in the same (unlicensed) spectrum. The real-time requirement for finding the optimal LTE resource allocation solution is on 1 ms time scale.

- **An Ultrafast GPU-based Proportional Fair Scheduler for 5G NR:** We study the popular proportional-fair (PF) scheduling problem in a 5G NR environment. The real-time requirement for determining the optimal resource allocation and modulation and coding scheme (MCS) for each user is 125 μs.

For the second type of problems, where there is no mathematical formulation, we propose to employ model-free deep learning (DL) or deep reinforcement learning (DRL) techniques along with judicious consideration of timing requirement throughout the design. Under this approach, we study the following two problems:

- **A DRL-based Approach to Dynamic eMBB/URLLC Multiplexing in 5G NR:** We study the problem of dynamic multiplexing of eMBB and URLLC on the same channel through preemptive resource puncturing. The real-time requirement for determining the optimal URLLC puncturing solution is 1 ms.
- **A DL-based Link Adaptation for eMBB/URLLC Multiplexing in 5G NR:** We investigate MCS selection for eMBB traffic under the impact of URLLC preemptive puncturing. The real-time requirement for determining the optimal MCSs for all eMBB transmissions scheduled in a transmission interval is 125 μs.

Dedication

To my beloved family.

Acknowledgments

First of all, I would like to express my utmost gratitude to my Ph.D. advisor, Prof. Tom Hou, the Bradley Distinguished Professor of ECE at Virginia Tech. I was fortunate to be admitted to Prof. Hou's group and given the opportunity to pursue a Ph.D. degree. Throughout my five years at Virginia Tech, Prof. Hou has been incredibly dedicated and supportive to my studies. He spent countless hours with me to discuss, define and develop research problems and solutions. He taught me how to conduct academic research at the highest quality. He also inspired me to be responsible, professional, and visionary for my future career beyond the Ph.D. stage. What I have learned the most from the numerous hours working with Prof. Hou are the academic writing skills to produce every sentence in the most precise and articulate way. Prof. Hou went through all my research papers with me line by line to improve the writing quality. It would be impossible for me to complete this dissertation and all my research works without the help and guidance from Prof. Hou. The experience of learning and working with Prof. Hou is most precious in my academic life.

My gratitude extends to Prof. Wenjing Lou, the co-director of the Complex Network and Security Research (CNSR) lab. During our annual CNSR workshops, Prof. Lou provided me many important suggestions on my research direction and topics, and have enlightened me to explore more possibilities in my Ph.D. study. The leadership of Prof. Hou and Prof. Lou has laid the foundation for the success of the CNSR members and alumni. I also want to thank all other members in my Ph.D. advisory committee for their time and effort: Prof. Jeffrey H. Reed, Prof. Wu-chun Feng, and Prof. Weijun Xie. Their comments and feedbacks have helped me improve the quality of this dissertation.

I would like to extend my thanks to all my current and former colleagues in the CNSR

Lab for their friendship, help and support: Xu Yuan, Xiaoqi Qin, Brian Jalaian, Qingyu Liu, Yongce Chen, Shaoran Li, Chengzhang Li, Yubo Wu, and Lei Li. Our group's alumni Xu, Xiaoqi and Brian gave me important guidance on my research journey at Virginia Tech. The discussion with Xu inspired me to explore my first research problem in my Ph.D. study (on fair coexistence between LTE and Wi-Fi). Xiaoqi taught me how to use the CPLEX optimizer to solve my first optimization problem. Brian worked together with me on a course project and we made it to publish the result as a journal paper. My labmates Yongce, Shaoran and Chengzhang have closely collaborated with me on many research publications. We often discussed about problems and solutions late into the nights. I also treasure the friendship with Qingyu (current post-doc at CNSR), Yubo and Lei, whom I will remember in my life. In particular, during the pandemic of COVID-19, the care and support among the CNSR members, especially that from Prof. Hou, helped everyone go through the difficult time well and safe.

Last but not least, I want to express my gratitude to my parents, Liping Lv and Qinghua Huang for giving me life and bringing me up. Your hard work and diligence ensured me a happy and positive childhood, which is the key to all my fulfillment in the adulthood. This dissertation is one of the utmost outcomes of your unconditional love. I also want to thank all other members of my family for their priceless love, understanding, and encouragement throughout my life.

Funding Acknowledgments

This research was supported in part by the National Science Foundation (NSF) under Grants CNS-1617634, CNS-1642873 and CNS-1800650, the Office of Naval Research (ONR) under Grant N00014-15-1-2926, Virginia Commonwealth Cyber Initiative (CCI), and the NVIDIA AI Lab (NVAIL) in Santa Clara, CA for its unrestricted gift and equipment donation.

Contents

List of Figures xviii

List of Tables xxii

1 Introduction 1

1.1 Motivation and Objectives 1

1.2 Dissertation Outline and Contributions 4

2 Real-Time Scheduling to Achieve Fair LTE/Wi-Fi Coexistence 8

2.1 Introduction 8

2.2 Related Work 12

2.3 System Architecture 13

2.4 Mathematical Modeling 19

2.5 CURT – A Novel Real-Time Scheduler 29

2.5.1 An Overview of CURT 30

2.5.2 Problem Decomposition 31

2.5.3 Feasibility Check of Sub-Problems 33

2.5.4 Computational Complexity 36

2.5.5 Guaranteeing Feasibility via Traffic Management 36

2.6 Implementation . . . 37
2.7 Validation . . . 42
2.7.1 Experimental Platform and Network Parameters . . . 42
2.7.2 Results . . . 45
2.8 Chapter Summary . . . 50

3 An Ultrafast GPU-based Proportional Fair Scheduler for 5G NR 52

3.1 Introduction . . . 52
3.2 Related Work . . . 56
3.3 A Primer on NR Air Interface . . . 58
3.4 A Formulation of the PF Scheduling Problem . . . 60
3.4.1 Modeling and Formulation . . . 60
3.4.2 PF Objective Function . . . 64
3.4.3 Problem Formulation . . . 65
3.5 The Real-Time Challenge . . . 67
3.6 A GPU-based Real-Time Scheduler Design . . . 69
3.6.1 Basic Idea and Roadmap . . . 69
3.6.2 Decomposition . . . 70
3.6.3 Search Intensification . . . 73
3.6.4 Sampling and Choosing the Best Solution . . . 81

3.7 Implementation 83
3.7.1 Why GPU 84
3.7.2 Overall Architecture 84
3.7.3 Some Details 86
3.8 Experimental Validation 89
3.8.1 Experiment Platform 89
3.8.2 Settings 89
3.8.3 Results 90
3.9 Chapter Summary 97

4 A DRL-based Approach to Dynamic eMBB/URLLC Multiplexing in 5G NR 98

4.1 Introduction 98
4.2 Dynamic Multiplexing of eMBB and URLLC 102
4.3 Related Work 105
4.4 Why Model-based Approaches Will Not Work 106
4.5 DEMUX: An Overview 108
4.5.1 Why DRL 108
4.5.2 Architecture of DEMUX 110
4.6 Design of the Scheduling Plane 113
4.6.1 An Example 113

4.6.2 Approximate Algorithm μ . . . 115
4.6.3 Guaranteeing Feasibility with g and f . . . 118
4.7 Design of the Learning Plane . . . 122
4.7.1 Challenge and Proposed Approach . . . 122
4.7.2 DDPG in a Nutshell . . . 124
4.7.3 Convergence Problem . . . 125
4.7.4 Our Design . . . 126
4.8 Implementation . . . 133
4.8.1 PHY-MAC NR Simulator . . . 133
4.8.2 DRL Module . . . 137
4.8.3 Putting Our Implementation to Work . . . 138
4.9 Validation . . . 140
4.9.1 Experimental Settings . . . 140
4.9.2 Benchmark Comparison . . . 143
4.9.3 Results . . . 144
4.10 Chapter Summary . . . 149

5 A DL-based Link Adaptation for eMBB/URLLC Multiplexing in 5G NR 151

5.1 Introduction . . . 151
5.2 Related Work . . . 158

5.2.1 eMBB/URLLC Multiplexing 159
5.2.2 DL for Wireless Communications and Networking 159
5.3 System Model and Problem Statement 160
5.4 DELUXE: Architecture and Key Ideas 166
5.5 Reducing Complexity of Input Data 167
5.5.1 Input Data 167
5.5.2 Transformation to Lower-Dimensional Representation 170
5.5.3 A Case Study 175
5.6 Learning and Prediction 177
5.7 Calibration and MCS Selection 183
5.8 Real-time Implementation 189
5.9 Performance Evaluation 192
5.9.1 Experimental Platform and Settings 192
5.9.2 A Case Study 196
5.9.3 Varying Network Parameters 198
5.10 Chapter Summary 202

6 Summary and Future Work **203**

6.1 Summary 203
6.2 Future Work 206

Bibliography 208

List of Figures

2.1 Coexistence of LTE and Wi-Fi in an area. 12

2.2 CSAT-based scheduling. 16

2.3 Radio resource arrangement in LTE and setting of scheduling frame. 18

2.4 Decomposition of OPT-R. 31

2.5 An illustration of our implementation of CURT. 37

2.6 Achieved objective value and timing performance of CURT. 44

2.7 Performance of CURT under increasing per-user rate requirement. Objective value of -1 indicates infeasibility. 47

2.8 Performance of CURT under increasing number of users. Objective value of -1 indicates infeasibility. 48

2.9 Performance of CURT under temporal channel variations for 30 users. User mobility $v = 1.5$ km/h. 49

2.10 Performance of CURT under temporal channel variations for 30 users. User mobility $v = 3.0$ km/h. 50

3.1 A frame structure of NR. 58

3.2 Spectral efficiencies of different MCSs ([36], Table 5.1.3.1-1). 63

3.3 $\bar{\beta}^j$ as a function of iterations j in Algorithm 2. 79

3.4 $\boldsymbol{v}_j$ after the j-th iteration in Algorithm 2. 80

3.5 CDF as a function of optimality gap (ϵ) based on $10,000$ samples. 82

3.6 A flow chart illustrating our implementation. 85

3.7 An illustration of parallel reduction. 88

3.8 GPF+'s performance vs. other state-of-the-art PF schedulers for $U = 25$ users. 91

3.9 GPF+'s performance vs. other state-of-the-art PF schedulers for $U = 50$ users. 93

3.10 GPF+'s performance vs. other state-of-the-art PF schedulers for $U = 75$ users. 94

3.11 GPF+'s performance vs. other state-of-the-art PF schedulers for $U = 100$ users. 95

4.1 An illustration of URLLC resource preemption. 103

4.2 Scheduling plane and learning plane in DEMUX. 110

4.3 Deep function approximator μ. 116

4.4 Relationship between the proportion of RBGs allocated to an eMBB user and its normalized expected PF utility. 128

4.5 An implementation diagram of our NR simulator. 134

4.6 Learning curves of DEMUX and original DDPG under different URLLC packet arrival rates. 143

4.7 An example of URLLC preemption by DEMUX. Resources allocated to each eMBB user are represented by a different color. Resources preempted by URLLC are in white blank. 145

4.8 Sum of utility comparison for 10 eMBB users under CDL and TDL channel models. 146

4.9 Cell throughput comparison for 10 eMBB users under CDL and TDL channel models. 147

4.10 Sum of utility comparison for 30 eMBB users under CDL and TDL channel models. 148

4.11 Cell throughput comparison for 30 eMBB users under CDL and TDL channel models. 149

5.1 An illustration of URLLC preemptive puncturing. 152

5.2 BLER and throughput performance of an eMBB link before and after the occurrence of URLLC puncturing under adaptive and non-adaptive MCS configurations. 155

5.3 An NR macro-cell with eMBB/URLLC multiplexing. 162

5.4 Spectral efficiencies of the 29 MCS levels defined in NR standard (Table 5.1.3.1-1 in [36]). 163

5.5 Architectural overview of DELUXE. This figure illustrates the MCS selection process for a given $e_i(t)$. 165

5.6 The change of information loss through the iterations. 176

5.7 Structure of DFA μ. 178

5.8 BLER performance of 10 eMBB users without calibrating μ's approximation error. 183

5.9 Concurrent forward propagation for all $e_i(t) \in \mathcal{E}(t)$. 191

5.10 BLER performance under DELUXE (a) and EESM (b). 195

5.11 Throughput comparison of DELUXE and EESM. 196

5.12 CDF of the MCS difference between EESM and DELUXE for each of the 10 users. 197

5.13 BLER and sum throughput performance under varying λ under DELUXE and EESM. Multi-path channel model is CDL-C. 199

5.14 BLER and sum throughput performance under varying λ under DELUXE and EESM. Multi-path channel model is TDL-C. 200

5.15 BLER and sum throughput performance under varying number of eMBB users under DELUXE and EESM. 200

List of Tables

2.1 Notation in Chapter 2 14

2.2 Classification of input to CURT. 39

3.1 OFDM numerologies in NR [2]. 53

3.2 Notation in Chapter 3 62

3.3 Breakdown of GPF's execution time consumed in different stages. Data format: (mean (μs), standard deviation). 96

4.1 Notation in Chapter 4 109

4.2 MCS levels and measured working SNRs (dB) under BLER = 0.1. 133

4.3 Key parameter settings in the the NR simulator. 141

4.4 Computation time of DEMUX. Results are averaged over 10^5 TTIs. 148

5.1 Notation in Chapter 5 161

5.2 Determine the CB length for $e_i(t)$. 168

5.3 Key network settings. 193

5.4 Execution time of DELUXE. Results are averaged over 10^5 TTIs. 202

Chapter 1

Introduction

1.1 Motivation and Objectives

Optimization of radio resource allocation in wireless networks are usually constrained by stringent real-time requirements. Such real-time requirements come from:

(i) *The physical properties of wireless channels, such as the channel coherence time [1].* Channel coherence time refers to the time duration over which the channel condition can be considered constant. For example, with a moving speed of 60 km/h, the coherence time on a 6 GHz channel is roughly 0.5 ms. An optimal resource allocation solution based on instantaneous channel conditions must be determined and applied to the network within the channel coherence time. Beyond the coherence time, the solution will be no longer efficient due to the variation of channel conditions.

(ii) *The small time resolution for resource allocation in modern cellular networks.* Resource allocation in 4G LTE or 5G NR is performed every transmission time interval (TTI), where a TTI is fixed to 1 ms in LTE and can be as short as 125 μs in NR under OFDM numerology 3 [2]. Resource allocation solution must be computed for each TTI with a computation time lower than the TTI duration.

(iii) *The ultra-low latency constraints from emerging applications.* 5G NR needs to support delay-sensitive applications that require an end-to-end latency on $\sim$1 ms level, such

as augmented/virtual reality [44, 45], autonomous vehicles [46] and smart grid [42]. Such use cases are termed as ultra-reliable and low-latency communications (URLLC) in 5G NR [35]. To meet such a latency requirement, TTI duration as well as the time resoulution for resource allocation must be less than 1 ms.

Although real-time resource optimization is critical for wireless network operations in the field, it has hardly been considered as a key constraint in the research community. Existing approaches to resource optimization can be classified into three categories:

(i) Exact or approximate optimization techniques from operations research such as the branch-and-bound (BB) methods (with cutting plane) [23], linear programming (LP) relaxations [8], and reformulation linearization techniques (RLT) [69] have been extensively used to find optimal or near-optimal solutions. The resource optimization problems in wireless networks are usually mathematically formulated into integer or mixed-integer optimization programs [24], which are NP-hard in general and may include tens of thousands of decision variables and constraints. The computational time of an optimal or provably near-optimal solution is usually prohibitively high, typically many orders of magnitude higher than the real-time requirements in wireless networks. Thus these methods can only be used for offline performance benchmarking. They are not useful for real-time network operations.

(ii) Due to the enormous computational complexity of the first approach, the research community has shown strong interest in developing fast heuristic solutions based on the unique features associated with each problem, either its physical properties or its mathematical structure. These heuristics are usually developed with the target of finding "good" solutions, instead of optimal or probably near-optimal solutions. The goal is to have a solution with asymptotically polynomial-time complexity (in "big

O" notation [30]). Although having polynomial-time complexities, these heuristics are typically not tested against wall-clock time, and can hardly meet real-time requirements on 1 ms or sub-1 ms time scale. For example, a greedy heuristic for resource allocation may go through millions of iterations (product of the numbers of resource units and users) to find a solution with a computational time of 10s or even 100s of ms.

(iii) Given the limitations of the above two approaches in the academic community, wireless engineers in the field have to resort to extremely simple solutions that can meet real-time requirement. These solutions are usually developed without focusing on an optimization objective. As a result, these solutions can only offer performance that is far from optimum (in terms of theoretically achievable objective value). An example of industry-grade real-time algorithms is the round-robin (RR) algorithm for resource scheduling in cellular networks [70]. In each TTI, RR assigns the resource blocks (RBs) to users in a cyclic manner regardless of the users' channel conditions, traffic demands, or throughput objectives. RR is attractive for its simplicity and low computation cost, but cannot achieve a desired optimization objective (e.g., maximizing spectral efficiency).

The objective of this dissertation is three-fold:

(i) We focus on resource optimization problems that arise from the industry, either from standardization efforts or operations of real-world wireless networks. This will ensure that our solutions can make an impact on real problems in the field.

(ii) We will develop solutions that can achieve optimal or near-optimal objective values. This will ensure the performance of our solutions is highly competitive against any designs from industry.

(iii) Our solutions must be able to meet the real-time requirement in the system. This objective sets us apart from most other existing approaches in the academic community.

1.2 Dissertation Outline and Contributions

This dissertation focuses on two types of resource optimization problems arising from wireless networks: i) problems that have closed-form mathematical models, and ii) problems that cannot be modeled in closed-form. For the first type of problems, we propose a novel approach that consists of:

(i) problem decomposition, which breaks an original optimization problem into a large number of small and independent sub-problems;

(ii) search intensification, which identifies the most promising problem sub-space and selects a small set of sub-problems to match the available GPU processing cores;

(iii) GPU-based large-scale parallel processing, which solves the selected sub-problems in parallel and finds a near-optimal solution to the original problem.

The efficacy of this approach has been illustrated by our solutions to the following two problems.

- **Real-Time Scheduling to Achieve Fair LTE/Wi-Fi Coexistence:** In Chapter 2, we investigate the fair coexistence between LTE and Wi-Fi in the unlicensed spectrum. This topic has drawn much attention from the industry in recent years (e.g., [3]). We study resource optimization for LTE with the objective of minimizing the adverse impact on co-channel Wi-Fi users. The real-time requirement for finding

an optimal solution is on ∼1 ms time scale. This problem involves the optimal division of transmission time for LTE and Wi-Fi across multiple unlicensed bands, and the resource allocation among LTE users within LTE's "ON" periods. We formulate this optimization problem as a closed-form mixed-integer linear program and prove its NP-hardness. Then by exploiting the unique problem structure, we design a real-time solution that is based on problem decomposition and GPU-based parallel processing techniques. Results from an implementation on the NVIDIA GPU/CUDA platform demonstrate that the proposed solution can achieve near-optimal objective and meet the 1 ms timing requirement.

- **An Ultrafast GPU-based Proportional Fair Scheduler for 5G NR:** In Chapter 3, we focus on the resource allocation for enhanced Mobile Broadband (eMBB) services in 5G NR networks. This problem involves the allocation of frequency-time resource blocks on an operating channel and the assignment of modulation and coding scheme (MCS) for each user [36]. In particular, we study the popular proportional-fair (PF) scheduling optimization problem in a 5G macro-cell. The real-time requirement for determining an optimal solution is 125 μs (under 5G numerology 3). We formulate this problem as an integer linear program. We present GPF+ — a GPU based real-time solution design. GPF+ first decomposes the original PF optimization problem into a large number of small and independent sub-problems. Then it uses a cross-entropy based search intensification technique to identify the most promising problem sub-space and select a small set of sub-problems to fit into a GPU. After solving the selected sub-problems in parallel using GPU cores, GPF+ finds the best sub-problem solution and uses it as the final scheduling solution. Experimental results show that GPF+ is able to provide near-optimal PF performance in a 5G cell while meeting the 125 μs real-time requirement.

For the second type of problems, there is no closed-form mathematical formulation (e.g., due to the intractability of analyzing the performance of PHY layer technologies in 5G NR networks). We propose to employ model-free deep learning (DL) or deep reinforcement learning (DRL) techniques, along with judicious consideration of timing requirement throughout the design. Under DL/DRL, we employ deep function approximators (neural networks) to learn the unknown objective function of an optimization problem, approximate an optimal algorithm to find resource allocation solutions, or discover important mapping functions related to the resource optimization. To meet the real-time requirement, we propose to augment DL or DRL methods with optimization techniques at the input or output of the deep function approximators to reduce their complexities and computational time. Under this approach, we study the following two problems:

- **A DRL-based Approach to Dynamic eMBB/URLLC Multiplexing in 5G NR:** In Chapter 4, we investigate the dynamic multiplexing of eMBB and URLLC services in 5G NR. This topic has been widely discussed within the 3GPP standards body [87]. Specifically, we study the problem of dynamic multiplexing of eMBB and URLLC on the same channel through preemptive resource puncturing. The real-time requirement for determining the optimal URLLC puncturing solution is 1 ms (under 5G numerology 0). A major challenge in solving this problem is that it cannot be modeled by a mathematical program. To address this issue, we develop a model-free DRL approach which employs a deep neural network to learn an optimal algorithm to allocate the URLLC puncturing over the operating channel, with the objective of minimizing the adverse impact from URLLC traffic on eMBB. Our contributions include a novel learning method that exploits the intrinsic properties of the URLLC puncturing optimization problem to achieve a fast and stable learning convergence, and a mechanism to ensure feasibility of the deep neural network's output puncturing

solution. Experimental results demonstrate that our DRL-based solution significantly outperforms state-of-the-art algorithms proposed in the literature and meets the 1 ms real-time requirement for dynamic multiplexing.

- **A DL-based Link Adaptation for eMBB/URLLC Multiplexing in 5G NR:** In Chapter 5, we focus on the eMBB side in the dynamic multiplexing and investigate the MCS selection for eMBB under the impact of URLLC preemptive puncturing. This problem is different from the URLLC puncturing scheduling problem we study in Chapter 4. MCS selection is an important mechanism under 5G standards [36] to ensure the reliability and spectral efficiency of each communication link. The real-time requirement for determining the optimal MCSs for all eMBB transmissions scheduled in a transmission interval is 125 μs (under 5G numerology 3). The objective is to have eMBB meet a given block-error rate (BLER) target under the adverse impact of URLLC puncturing. Since this problem cannot be modeled as a mathematical program, we propose a DL-based solution design that uses a deep neural network to learn and predict the BLERs of a transmission under each MCS level. Then based on the BLER predictions, an optimal MCS can be found for each transmission that can achieve the BLER target. To meet the real-time requirement, we implement this design through a hybrid CPU and GPU architecture to minimize the execution time. Experimental results show that our design can select optimal MCS under the impact of preemptive puncturing and meet the 125 μs timing requirement.

Chapter 2

Real-Time Scheduling to Achieve Fair LTE/Wi-Fi Coexistence

2.1 Introduction

There is a strong interest from cellular carriers to use existing unlicensed spectrum (e.g., the 5 GHz UNII bands) to boost cellular services. This approach is appealing for a number of reasons: (i) unlicensed spectrum is free (no need of auction and a license fee), (ii) the underlying bandwidth is substantial (e.g., 775 MHz available bandwidth in 5 GHz UNII bands), (iii) coexisting with other unlicensed wireless technologies (e.g., Wi-Fi) bears significantly fewer operational risk concerns when compared to sharing spectrum on the military bands (e.g., with radar systems). As a result, there have been significant activities on coexistence of cellular (LTE) and Wi-Fi in unlicensed bands from both industry [3, 4, 5, 6, 7] and academia [9, 10, 11, 12, 13, 14, 15, 16].

A key consideration in the design and operation of LTE in unlicensed band is to ensure fairness when they coexist with Wi-Fi. LTE was originally designed to work exclusively in operator-owned licensed bands. Its transmissions are centrally controlled and have no consideration for cross-technology coexistence [6]. In contrast, Wi-Fi employs CSMA/CA and is based on distributed contention. It can only transmit after the operating channel is clear and the lapse of its backoff period. Such incompatibility makes Wi-Fi highly vulnerable

to the presence of LTE in the same band.

To address this issue, a number of mechanisms have been proposed for LTE in unlicensed band, such as Listen-Before-Talk (LBT) [5, 7] and Carrier-Sensing Adaptive Transmission (CSAT) [3, 4]. LBT is a random access approach similar to Wi-Fi's CSMA/CA, while CSAT is based on centralized scheduling, which is native to LTE's operation. With proper design, both CSAT and LBT can achieve fair spectrum sharing between LTE and Wi-Fi. Although CSAT may cause collisions to Wi-Fi's on-going packets, such impact can be mitigated by configuring longer duration for each LTE transmission burst [11]. CSAT is fully compatible with 3GPP Release 10/11 and does not require any change of LTE specifications [10]. It can be quickly launched in countries that do not mandate implementing LBT (e.g., the US and China). Due to these benefits, operators such as T-Mobile have started supporting CSAT-based LTE-U in a number of US cities [17].

In this chapter, we consider the CSAT mechanism and study a scheduling problem for the coexistence of LTE and Wi-Fi in 5 GHz unlicensed spectrum. In this spectrum, there are multiple channels that can be used by LTE simultaneously. Under CSAT, the air time of each channel is divided into periodic LTE "on/off" cycles. The "on" and "off" periods are used by LTE and Wi-Fi for channel access, respectively. Optimal division of "on" and "off" periods is determined by the LTE eNodeB (eNB) based on Wi-Fi's traffic load as measured from carrier sensing. Within LTE's "on" period, the channel bandwidth is expanded into a group of sub-channels and it is at this level that the so-called resource blocks (RBs) are allocated to LTE users. Suppose there is a different number of Wi-Fi users on each channel. To support a set of LTE users across these channels, where each user has its own uplink (UL) and downlink (DL) rate requirements in unlicensed spectrum, the problem becomes how to perform radio resource allocation such that the impact of LTE on Wi-Fi is minimized.

We formulate the above scheduling problem for LTE/Wi-Fi coexistence as an optimiza-

tion problem. We will show that this problem is NP-hard, meaning that there is no efficient algorithm to obtain an optimal solution in real-time under practical network settings. Therefore, it is necessary to pursue a heuristic solution that can achieve near-optimal objective. The main challenge we need to address is that the scheduling solution must be obtained in real-time — with a computational time of $\sim$1 ms. This timing requirement comes from the fact that channel coherence time in 5 GHz bands is at most 10s of ms, meaning that a channel-dependent scheduling solution can remain valid only for 10s of ms. If the computation time is beyond this time limit, the solution would not be considered good since channel conditions may have already changed considerably.

The goal of this chapter is to design a scheduler that can find a near-optimal solution in $\sim$1 ms under practical LTE small cell settings. We propose CURT, which arises from either the abbreviation of CSAT based Unlicensed LTE Real-Time resource scheduling (from coexistence scheme's perspective) or CUDA-based Real-Time resource scheduling (from implementation's perspective). We summarize the main contributions of CURT as follows:

- For LTE scheduling, we consider a wide range of parameters in our scheduling problem so as to best resemble what one would encounter in the field. These include (i) multiple channels available for LTE/Wi-Fi coexistence; (ii) both UL and DL rate requirements of LTE users in unlicensed spectrum; (iii) variation of channel conditions across subchannels. We formulate this scheduling problem into an optimization problem with the objective of minimizing the adverse impact on Wi-Fi while meeting LTE users' rate requirements. Further, we prove that the above scheduling problem is NP-hard.

- We present CURT, a novel GPU-based scheduler design that can achieve near-optimal performance while meeting the real-time requirement on $\sim$1 ms time scale. In our design, by exploiting the unique problem structure, we decompose the original scheduling

problem into a large number of independent sub-problems encompassing all possible cases of parameter settings. Then by performing a simple and fast evaluation of the feasibility of each sub-problem in parallel through massive GPU processing cores, we can determine a near-optimal (or optimal) solution to the original problem.

- To validate the performance of CURT, we implement it using off-the-shelf NVIDIA Quadro P6000 GPUs in an integrated host-GPU architecture. Our implementation is based on meticulous considerations of GPU/CUDA architecture, mathematical structure of our proposed solution, and most importantly, the ∼1 ms constraint for overall scheduling time. Extensive experimental results confirm that CURT can consistently find near-optimal scheduling solution in ∼1 ms under practical LTE small cell settings. (e.g., 20 users per cell following 3GPP's evaluation methodology [5]) and meet all of our design objectives. This represents the first known CSAT-based scheduler design that can achieve real-time and near-optimal scheduling for LTE/Wi-Fi coexistence.

The remainder of this chapter is organized as follows. In Section 2.2, we review related work on LTE/Wi-Fi coexistence in unlicensed bands. In Section 2.3, we describe in detail the system architecture of CSAT-based LTE and state the underlying scheduling problem. In Section 2.4, we formulate the scheduling problem for LTE/Wi-Fi coexistence and prove its NP-hardness. In Section 2.5, we present CURT, a real-time scheduler for LTE in unlicensed band. In Section 2.6, we show how CURT is implemented on off-the-shelf NVIDIA Quadro P6000 GPUs. In Section 2.7, we conduct experiments to validate the performance of CURT. Section 2.8 concludes this chapter.

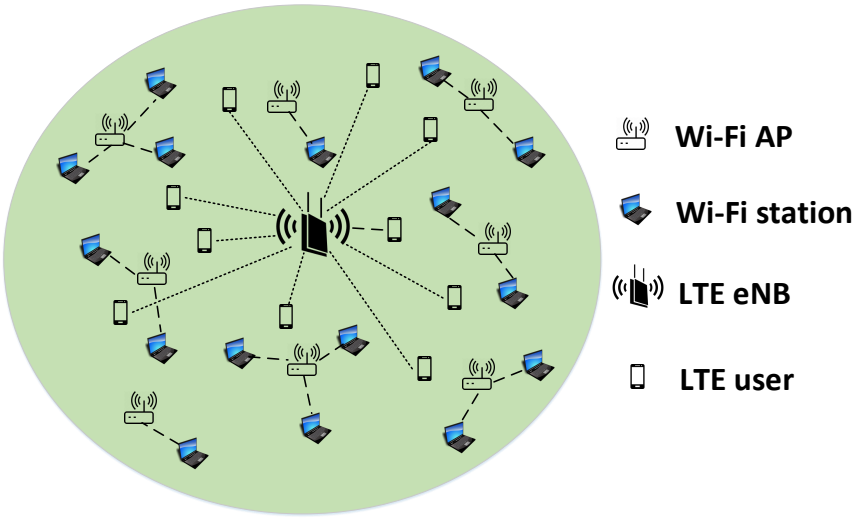


Figure 2.1: Coexistence of LTE and Wi-Fi in an area.

2.2 Related Work

In the research community, there have been a number of studies on LTE/Wi-Fi coexistence, such as modeling and analysis of LTE's impact on Wi-Fi [12, 13], optimizations of coexistence mechanisms [9, 15], and radio resource management of LTE in unlicensed band [14, 16].

In [12], Abdelfattah *et al.* developed an analytical model for Wi-Fi's collision probability and throughput when coexisting with CSAT-based LTE. In [13], Voicu *et al.* proposed a general framework to evaluate the performance of multiple technologies operating in the same unlicensed bands. Both [12] and [13] focused on modeling of LTE/Wi-Fi coexistence and did not address allocation of radio resources.

In [15], the authors derived optimal division of LTE "on" and "off" periods under CSAT for a given number of LTE and Wi-Fi users. In [9], the authors considered joint optimization of channel selection and CSAT parameters. A fairness criterion was derived for LTE and Wi-Fi sharing multiple unlicensed channels. The criterion requires LTE not to impact Wi-Fi more than another Wi-Fi network with the same traffic load. The efforts in [15] and [9]

addressed optimizations of CSAT parameters, but fell short to address resource management for LTE at the RB level.

In [14], Chen *et al.* addressed optimization of energy efficiency for CSAT-based LTE by studying RB allocation over licensed and unlicensed bands. The analysis in this work, however, did not consider channel fading effect on each individual RB, which is what will happen in practice. In [16], channel selection and per-frame RB scheduling were studied for coexistence between LTE and WLAN on multiple channels from unlicensed spectrum. An optimization problem was formulated with the objective of maximizing LTE's throughput while maintaining fairness between LTE and WLAN. The sequential algorithm proposed in this chapter, although being polynomial-time, involves a large amount of iterative computations for scheduling in each frame. For both [14] and [16], it is not clear if the proposed scheduling algorithms can meet real time requirement (i.e., $\sim$1 ms).

2.3 System Architecture

Taking into account regional regulations on transmit power in unlicensed bands [5], we consider small cell deployments of unlicensed LTE where an LTE small cell overlaps with multiple Wi-Fi APs, as shown in Fig. 2.1. Table 2.1 lists notation in this chapter. In Fig. 2.1, a set of LTE users $\mathcal{K}$ is served by a single LTE eNB while each Wi-Fi user is served by a nearby Wi-Fi AP.[1] Note that Fig. 2.1 only shows Wi-Fi nodes that fall in the LTE eNB's interference range. Potential Wi-Fi nodes outside the LTE eNB's interference range that are "hidden" from the eNB are not shown in this figure. The reason why we do not consider those hidden Wi-Fi nodes (outside the interference range of the eNB) is the following. Although those hidden Wi-Fi nodes may have adverse impact on Wi-Fi nodes that are inside the LTE

[1]The set $\mathcal{K}$ of LTE users that are offloaded to unlicensed bands can be determined by the carrier load balancing function defined in the LTE specification [18].

Table 2.1: Notation in Chapter 2

Symbol	Definition
$\mathcal{F}$	A set of channels shared between LTE and Wi-Fi
$\mathcal{S}_i$	A set of sub-channels on channel i
(i,j)	The jth sub-channel in $\mathcal{S}_i$
$\mathcal{K}$	A set of LTE users offloaded to unlicensed bands
T_0	Duration of a TTI
T_{SF}	Duration of a scheduling frame
N_{SF}	The number of TTIs in a scheduling frame
M	The number of radio frames in a scheduling frame
I_i^{UL}	Binary variable indicating whether or not channel i is selected for UL transmission
I_i^{DL}	Binary variable indicating whether or not channel i is selected for DL transmission
$n_{(i,j)}^{k,\mathrm{UL}}$	Integer variable denoting the amount of TRBs allocated to user k for UL transmission on sub-channel (i,j)
$n_{(i,j)}^{k,\mathrm{DL}}$	Integer variable denoting the amount of TRBs allocated to user k for DL transmission on sub-channel (i,j)
$n_{i,\max}^{\mathrm{UL}}$	The amount of TRBs reserved for LTE's UL transmission on channel i
$n_{i,\max}^{\mathrm{DL}}$	The amount of TRBs reserved for LTE's DL transmission on channel i
$C_{(i,j)}^{k,\mathrm{UL}}$	The UL achievable rate of user k on sub-channel (i,j)
$C_{(i,j)}^{k,\mathrm{DL}}$	The DL achievable rate of user k on sub-channel (i,j)
$R^{k,\mathrm{UL}}$	UL rate requirement of user k in unlicensed bands
$R^{k,\mathrm{DL}}$	DL rate requirement of user k in unlicensed bands
Q_i	The maximum number of TTIs that can be used for LTE scheduling on channel i
U_i	The number of Wi-Fi nodes on channel i
w_i	The weight reflecting the Wi-Fi traffic load on channel i
z	Objective value in Problem OPT-R

eNB's interference range, they are not of concern of LTE scheduling and thus is not part of our problem formulation. Further, in practice, the LTE eNB has no mechanism to detect such hidden Wi-Fi nodes outside its interference range. Each LTE user $k \in \mathcal{K}$ has both UL and DL rate requirements in the unlicensed spectrum, which are denoted by $R^{k,\mathrm{UL}}$ and $R^{k,\mathrm{DL}}$, respectively. The rate requirements should be configured dynamically by a traffic management mechanism in unlicensed bands as described in Section 2.5.5.

Suppose there is a number of channels in the unlicensed band that can be used by both LTE and Wi-Fi networks. Due to dense Wi-Fi deployment, there may not be enough clear channels for LTE and thus LTE has to coexist with Wi-Fi on some of these channels. In this chapter, we focus on this subset of channels, denoted by $\mathcal{F}$, where both LTE and Wi-Fi are present. For LTE, its transmission scheduling is centrally controlled by the eNB, and it can combine multiple channels for UL and DL transmissions via FDD carrier aggregation (CA).[2] Every channel $i \in \mathcal{F}$ (used for either UL or DL) is further divided into a set of sub-channels $\mathcal{S}_i$. Thus the frequency granularity of LTE is on sub-channel level. In contrast, for Wi-Fi, the frequency granularity is on the channel level, where an AP or station typically occupies the entire bandwidth of a channel (instead of a sub-channel) for DL or UL data transmission. Denote U_i as the number of Wi-Fi nodes on channel $i \in \mathcal{F}$.

CSAT-based LTE Scheduling We employ CSAT for LTE scheduling [4]. As shown in Fig. 2.2, CSAT is a time division multiplexing (TDM)-like channel access mechanism where each CSAT cycle (a.k.a a scheduling frame) consists of an LTE "off" and "on" period. During the "off" period, LTE transmission is suspended so that co-channel Wi-Fi nodes can access the medium. Once the "off" period is over, LTE network starts transmission regardless of channel status (even there is a collision). Since Wi-Fi senses the channel before a new transmission, it will cease to transmit during LTE's "on" period. In a CSAT cycle, the

[2]Under CA, only up to 5 carriers are possible [19]. Although there are more than 10 channels in 2.4 and 5 GHz bands, LTE can only select no more than 5 channels for coexistence. Thus we have $|\mathcal{F}| \leq 5$.

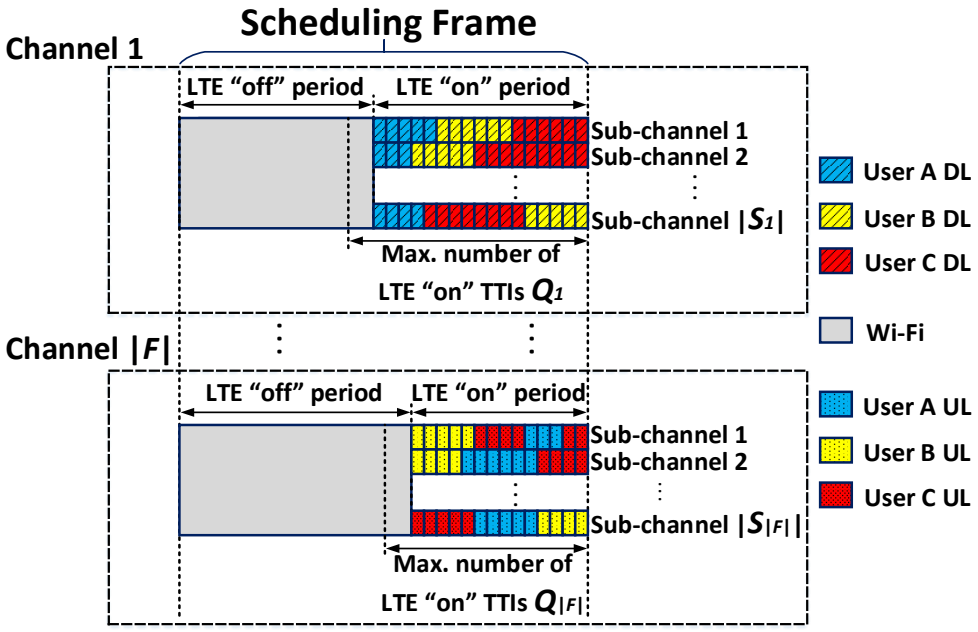


Figure 2.2: CSAT-based scheduling.

division of "off" and "on" periods on a channel is part of the scheduling problem. Through carrier sensing, the eNB measures Wi-Fi traffic load on each channel, and uses it as input to determine "off" and "on" periods on this channel.

Radio Resource Arrangement in LTE An illustration of LTE's radio resource arrangement is given in Fig. 2.3. Specifically, on each channel, radio resource is organized as a two-dimensional resource grid [6]. In frequency domain, the channel is divided into a set of sub-channels, each with a bandwidth of 180 kHz. In time domain, we have consecutive *radio frames*, each with a duration of 10 ms. A radio frame consists of 10 sub-frames. The duration of a sub-frame is 1 ms, which is termed a *Transmission Time Interval* (TTI). A TTI is further divided into two time slots, each with a duration of 0.5 ms. A resource block (RB) is defined as a time-frequency resource unit with 180 kHz in frequency (a sub-channel) and 0.5 ms in time (a time slot). The time resolution for LTE scheduling is two consecutive RBs in a sub-frame, which we call a *Twin RBs* (TRB). Since each TRB is of 1 ms, a radio

frame consists of 10 TRBs.

Scheduling Frame and Coherence Time We define a scheduling frame (SF) as a consecutive M radio frames (refer to Fig. 2.3). Since a radio frame is 10 ms, the duration of a SF, denoted by T_{SF}, is equal to $10M$ ms.

The maximum number of radio frames that can be packed into a scheduling frame, M, is upper limited by the coherence time of the underlying channel. That is, M should be small enough so that there is no significant change of LTE users' channel conditions (as well as their achievable data rates) over a period of T_{SF}. As an example, consider the 5 GHz spectrum for an indoor deployment scenario. The channel coherence time T_{C} can be calculated by $T_{\text{C}} = \sqrt{\frac{9}{16\pi f_{\text{M}}^2}}$ [1], where $f_{\text{M}} = v/\lambda$ denotes the maximum Doppler shift, v is the user speed, and λ is the carrier wavelength. In an indoor small cell, assuming a user speed of 3 km/h [5], the coherence time on 5 GHz spectrum is $T_{\text{C}} = 30.58$ ms. Therefore, the maximum value M can take is 3 ($\leq 30.58/10$). That is, $T_{\text{SF}} = 30$ ms.

Denote the number of TTIs in a SF by N_{SF}. By definition, we have $N_{\text{SF}} = \frac{T_{\text{SF}}}{T_0} = 10M$, where $T_0 = 1$ ms denotes the duration of a TTI. Within a CSAT "on/off" cycle, we will have an integral number of TTIs for both "on" and "off" periods. Further, we assume perfect time synchronization so that the boundaries of TTIs and SFs across all channels occupied by LTE are perfectly aligned.

Problem Statement We are interested in addressing the following problem: *Given that a set $\mathcal{K}$ of LTE users are to coexist with Wi-Fi, how do we minimize LTE traffic's adverse impact on Wi-Fi users while meeting each LTE user's UL and DL rate requirements?* To answer this question, we must address the following sub-problems:

- (i) For LTE, since each user has both UL and DL data traffic, we must decide how to use each channel in $\mathcal{F}$. That is, should a channel $i \in \mathcal{F}$ be used for UL or DL

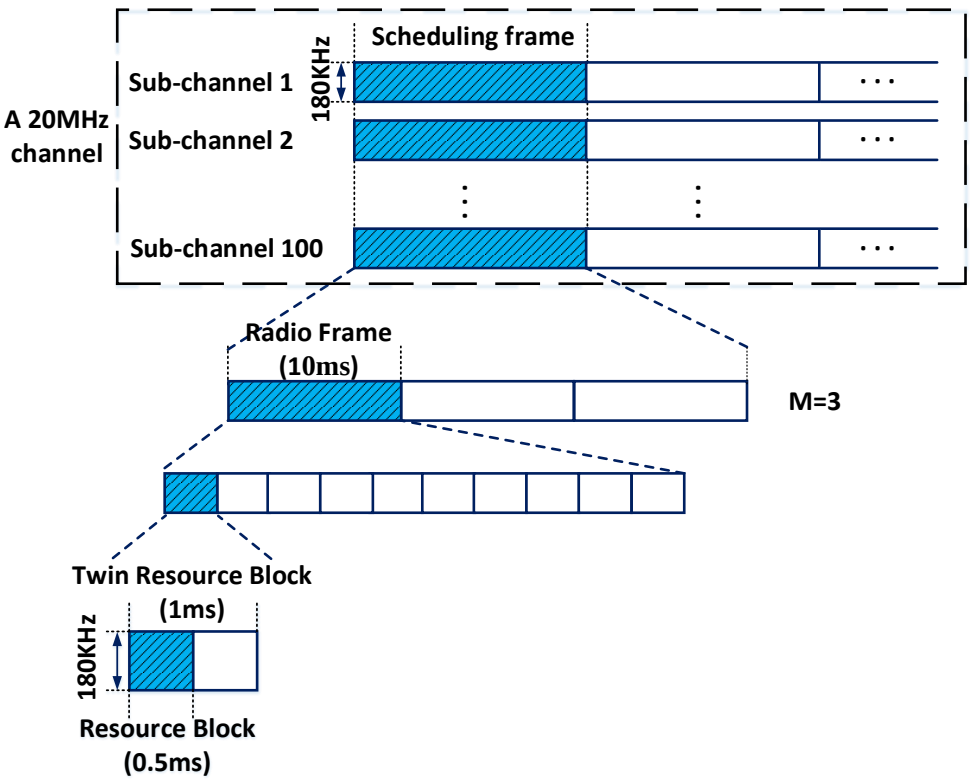


Figure 2.3: Radio resource arrangement in LTE and setting of scheduling frame.

transmission?

- (ii) For the coexistence of LTE and Wi-Fi on each channel in $\mathcal{F}$, we must decide how to divide each SF into "off" and "on" periods. The adverse impact on Wi-Fi depends on the length of "on" period. Then our objective is to minimize such adverse impact across all channels through optimally assigning "off" and "on" periods on each channel.

- (iii) To meet the UL and DL rate requirements of each LTE user, we need to properly allocate the TRBs on each sub-channel. Each user can be allocated to TRBs from multiple channels (and sub-channels). The optimal allocation of TRBs is not trivial because the achievable data rate of each user varies across different sub-channels (due to frequency-selective channel fading).

- (iv) Last but perhaps most significant is that we are interested in designing a *real-time* solution. This real-time requirement means that the scheduling solution must be found within the "off" period of each SF (across all channels in $\mathcal{F}$). This will ensure that LTE users can follow the pre-computed, optimized transmission schedule in "on" periods on all channels. Given that T_{SF} is typically several 10s of ms and optimal "off" periods may be less than 10 ms, the scheduling time must be within a few ms. In this chapter, we use 1 ms as our target scheduling time.

2.4 Mathematical Modeling

In this section, we develop a mathematical model for the resource scheduling problem for LTE/Wi-Fi coexistence.

UL/DL Channel Assignment Referring to Fig. 2.2, consider the set of channels $\mathcal{F}$ where each channel is shared between LTE and Wi-Fi users. For LTE, denote I_i^{UL} and I_i^{DL}

as binary variables to indicate whether channel $i \in \mathcal{F}$ is used for UL and DL transmissions, respectively, i.e.,

$$I_i^{\mathrm{UL}} = \begin{cases} 1, & \text{if channel } i \in \mathcal{F} \text{ is selected for UL;} \\ 0, & \text{otherwise.} \end{cases}$$

$$I_i^{\mathrm{DL}} = \begin{cases} 1, & \text{if channel } i \in \mathcal{F} \text{ is selected for DL;} \\ 0, & \text{otherwise.} \end{cases}$$

Since each channel can only be used by LTE for either UL or DL transmission, but not both, we have:

$$I_i^{\mathrm{UL}} + I_i^{\mathrm{DL}} \leq 1 \quad (i \in \mathcal{F}). \tag{2.1}$$

Effective Occupancy by LTE on A Channel Referring to Fig. 2.2, for each channel $i \in \mathcal{F}$, there is a set of sub-channels $\mathcal{S}_i$. Scheduling for LTE is performed on sub-channel level. On each sub-channel of channel i, LTE's transmission time (either UL or DL) may not terminate at the same time. Denote $(i,j) \in \mathcal{S}_i$ as sub-channel j on channel i. Then as far as Wi-Fi is concerned, channel i is available only if LTE ceases transmissions on *all* sub-channels.

To model this effective channel occupancy by LTE, denote $n_{(i,j)}^{k,\mathrm{UL}}$ and $n_{(i,j)}^{k,\mathrm{DL}}$ as the number of TRBs on sub-channel $(i,j) \in \mathcal{S}_i$ within a SF that are allocated to user $k \in \mathcal{K}$ for UL and DL transmissions, respectively. If channel i is selected for UL transmission (i.e., $I_i^{\mathrm{UL}} = 1$), then LTE's usage of TTIs on sub-channel (i,j) across all users in $\mathcal{K}$ is $\sum_{k\in\mathcal{K}} n_{(i,j)}^{k,\mathrm{UL}}$. Denote $n_{i,\max}^{\mathrm{UL}}$ as the effective channel occupancy by LTE on channel i across all $|\mathcal{S}_i|$ sub-channels. Then,

$$n_{i,\max}^{\mathrm{UL}} = \max_{j\in\mathcal{S}_i} \sum_{k\in\mathcal{K}} n_{(i,j)}^{k,\mathrm{UL}} \quad (i \in \mathcal{F}). \tag{2.2}$$

Likewise, if channel i is selected for DL transmission (i.e., $I_i^{\mathrm{DL}} = 1$), then LTE's usage of

TTIs on sub-channel (i, j) across all users in $\mathcal{K}$ is $\sum_{k\in\mathcal{K}} n_{(i,j)}^{k,\mathrm{DL}}$. Denote $n_{i,\max}^{\mathrm{DL}}$ as the effective channel occupancy by LTE on channel i. We have:

$$n_{i,\max}^{\mathrm{DL}} = \max_{j\in\mathcal{S}_i} \sum_{k\in\mathcal{K}} n_{(i,j)}^{k,\mathrm{DL}} \quad (i \in \mathcal{F}). \tag{2.3}$$

Within a SF on channel i, the usable time duration (in unit of TTIs) for LTE is determined by $n_{i,\max}^{\mathrm{UL}}$ (if $I_i^{\mathrm{UL}} = 1$) or $n_{i,\max}^{\mathrm{DL}}$ (if $I_i^{\mathrm{DL}} = 1$). While the time duration left for Wi-Fi is $N_{\mathrm{SF}} - n_{i,\max}^{\mathrm{UL}}$ (for UL) or $N_{\mathrm{SF}} - n_{i,\max}^{\mathrm{DL}}$ (for DL) TTIs.

Upper Bound on LTE Usage To ensure that LTE does not monopolize radio resource of each channel $i \in \mathcal{F}$, it is important to set up an upper bound on LTE's transmission time for its "on" period on each channel [5]. Let Q_i ($Q_i < N_{\mathrm{SF}}$) denote the upper bound on the number of TTIs that LTE can use for UL or DL transmission on channel i within a SF. Then

$$n_{i,\max}^{\mathrm{UL}} \le I_i^{\mathrm{UL}} Q_i \quad (i \in \mathcal{F}), \tag{2.4}$$

$$n_{i,\max}^{\mathrm{DL}} \le I_i^{\mathrm{DL}} Q_i \quad (i \in \mathcal{F}). \tag{2.5}$$

The setting of Q_i's depends on the fairness criterion considered for LTE/Wi-Fi coexistence. For example, a popular fairness criterion is that on each channel, LTE should not impact Wi-Fi more than another Wi-Fi network with the same traffic load [5, 9]. With the assumption that both LTE and Wi-Fi users have full-buffered data traffic, a reasonable setting based on such criterion is to let $Q_i = \left\lfloor N_{\mathrm{SF}} \frac{|\mathcal{K}|/|\mathcal{F}|}{|\mathcal{K}|/|\mathcal{F}|+U_i} \right\rfloor$, where $|\mathcal{K}|/|\mathcal{F}|$ is the number of LTE users per channel and U_i is the number of Wi-Fi nodes on channel i. Basically, this setting allocates transmission time to LTE and Wi-Fi in proportion to the number of nodes per channel in each network, and thus can be considered fair to both LTE and Wi-Fi.

For practical implementation of the scheduling algorithm, one could employ a different

fairness criterion and optimize Q_i's accordingly. Then fairness to Wi-Fi is ensured when solutions feasible to constraints (2.4) and (2.5) are used for scheduling LTE transmissions. Since Q_i's are just input parameters to the scheduling problem, how Q_i's are determined will not affect the solution design. Our proposed solution algorithm works under any setting of Q_i's based on the fairness criterion chosen by the operator. In fact, it is allowed to adjust Q_i's per SF, if it is deemed necessary.

Meeting LTE User Rate Requirement For each user $k \in \mathcal{K}$, to ensure both of its UL and DL rate requirements are met, we have the following constraints:

$$R^{k,\mathrm{UL}} \leq \frac{\sum_{i\in\mathcal{F}} \sum_{j\in\mathcal{S}_i} n^{k,\mathrm{UL}}_{(i,j)} C^{k,\mathrm{UL}}_{(i,j)} T_0}{T_{\mathrm{SF}}} \quad (k \in \mathcal{K}), \tag{2.6}$$

$$R^{k,\mathrm{DL}} \leq \frac{\sum_{i\in\mathcal{F}} \sum_{j\in\mathcal{S}_i} n^{k,\mathrm{DL}}_{(i,j)} C^{k,\mathrm{DL}}_{(i,j)} T_0}{T_{\mathrm{SF}}} \quad (k \in \mathcal{K}), \tag{2.7}$$

where $C^{k,\mathrm{UL}}_{(i,j)}$ and $C^{k,\mathrm{DL}}_{(i,j)}$ are UL and DL achievable data rates for user k on sub-channel (i,j), respectively. The data rates $C^{k,\mathrm{UL}}_{(i,j)}$'s and $C^{k,\mathrm{DL}}_{(i,j)}$'s are obtained based on users' CSI reports [7]. This is how it is done in real-world FDD LTE systems. During the "on" period when LTE is transmitting, if there is any interference from Wi-Fi, then such interference will be captured in the CSI reports as well as the estimated parameters $C^{k,\mathrm{UL}}_{(i,j)}$'s and $C^{k,\mathrm{DL}}_{(i,j)}$'s. That is, $C^{k,\mathrm{UL}}_{(i,j)}$'s and $C^{k,\mathrm{DL}}_{(i,j)}$'s are obtained via channel estimation and have already considered interference from Wi-Fi, if there is any. By the definition of SF in Section 2.3, $C^{k,\mathrm{UL}}_{(i,j)}$'s and $C^{k,\mathrm{DL}}_{(i,j)}$'s remain constant during each SF.

Objective Function and Problem Formulation In a SF, the transmission time of LTE on channel $i \in \mathcal{F}$ is determined by $I^{\mathrm{UL}}_i \cdot n^{\mathrm{UL}}_{i,\max} + I^{\mathrm{DL}}_i \cdot n^{\mathrm{DL}}_{i,\max}$. For the same transmission time duration, LTE's impact on Wi-Fi depends on the traffic load of Wi-Fi. The heavier traffic that is being served by Wi-Fi on the same channel, the greater the impact of LTE

on Wi-Fi. To take this into consideration, we introduce a weight parameter w_i to reflect Wi-Fi's traffic load on channel $i \in \mathcal{F}$. A simple example is to set w_i to the number of Wi-Fi nodes on channel i, i.e., $w_i = U_i$. To find U_i on each channel, the eNB can monitor Wi-Fi's channel usage during "off" periods, which are allocated to Wi-Fi transmission. For example, with the methods proposed in [20, 21], U_i can be determined online based on the proportion of observed busy time slots. This is feasible since an LTE eNB using unlicensed band is expected to be able to perform carrier sensing [3]. For a given w_i, the impact of LTE on Wi-Fi on channel i can be quantitatively measured by $w_i(I_i^{\mathrm{UL}} \cdot n_{i,\max}^{\mathrm{UL}} + I_i^{\mathrm{DL}} \cdot n_{i,\max}^{\mathrm{DL}})$. Note that the weight w_i is the same for uplink and downlink because regardless of the direction (uplink or downlink) in which LTE transmits, the worst-case lost transmission time for Wi-Fi is equal to the duration of LTE's "on" period.

The reason why we use the duration of LTE "on" period weighted by the number of Wi-Fi nodes on the channel to model the impact of LTE on Wi-Fi is as follows. In practice, there is no channel training and coordination mechanism between LTE and Wi-Fi. As a result, the centralized LTE scheduler (located at eNB) has no information on channel statistics of radio links to Wi-Fi (e.g., path loss and fast-fading coefficients), the received interference power level at Wi-Fi, or the traffic condition of Wi-Fi. Thus in LTE scheduling optimization, one cannot assume such information is available. To have a reasonable model of the impact of LTE on Wi-Fi, we use the length of LTE's "on" period in each scheduling frame, during which Wi-Fi nodes are not expected to transmit. Although Wi-Fi may transmit opportunistically when fast deep fading occurs on the channel during "on" period, such information is not available at the LTE scheduler. Therefore, the "on" period is actually the worst-case loss of transmission period for Wi-Fi. In addition, during an "on" period, the impact of LTE on Wi-Fi depends on the traffic load of Wi-Fi on the same channel, i.e., the higher the Wi-Fi traffic load, the more severe the impact. As it is impossible for the LTE scheduler to know

the actual Wi-Fi traffic load, one can only use the number of active Wi-Fi nodes through sensing the channel as a load indicator, assuming the traffic at Wi-Fi nodes is persistent. Previous work has shown the feasibility of obtaining such information through carrier sensing methods [20, 21].

Since LTE's impact on Wi-Fi varies from channel to channel, a plausible objective for the network operator is to minimize the maximum of LTE's impact across all channels, i.e.,

$$\min \max_{i \in \mathcal{F}} w_i (I_i^{\mathrm{UL}} \cdot n_{i,\max}^{\mathrm{UL}} + I_i^{\mathrm{DL}} \cdot n_{i,\max}^{\mathrm{DL}}). \tag{2.8}$$

This is the objective we use in this chapter.

Our optimization problem can be formally stated as follows:

OPT

$$\begin{aligned}
\text{minimize} \quad & \max_{i \in \mathcal{F}} w_i (I_i^{\mathrm{UL}} \cdot n_{i,\max}^{\mathrm{UL}} + I_i^{\mathrm{DL}} \cdot n_{i,\max}^{\mathrm{DL}}) \\
\text{subject to} \quad & \text{UL/DL channel assignment: (2.1),} \\
& \text{Effective channel occupancy by LTE: (2.2), (2.3),} \\
& \text{Upper bound on LTE usage: (2.4), (2.5),} \\
& \text{Per-user rate requirement: (2.6), (2.7),} \\
& n_{i,\max}^{\mathrm{UL}}, n_{i,\max}^{\mathrm{DL}}, n_{(i,j)}^{k,\mathrm{UL}}, n_{(i,j)}^{k,\mathrm{DL}} \in \mathbb{N}, \\
& I_i^{\mathrm{UL}}, I_i^{\mathrm{DL}} \in \{0,1\} \quad (i \in \mathcal{F}, j \in \mathcal{S}_i, k \in \mathcal{K}).
\end{aligned}$$

The solution to OPT determines the allocation of resources for LTE and Wi-Fi within an entire SF (CSAT cycle). When implementing the solution in an LTE small cell, resource allocated to LTE users can be delivered in the same per-TTI manner as in licensed band operation. Specifically, the solution for an entire SF is stored in the eNB, and in each TTI

for LTE's transmission, the eNB informs users of their resource allocation for the current TTI via control channel signaling.

A Reformulation In Problem OPT, since the objective function involves integer variables and two levels of max functions (due to (2.2) and (2.3)), a reformulation would be needed. In particular, in the presence of constraints (2.4) and (2.5), the objective function can be simplified to $\max_{i\in\mathcal{F}} w_i(n^{\mathrm{UL}}_{i,\max}+n^{\mathrm{DL}}_{i,\max})$. To remove the two levels of max functions, we define $z = \max_{i\in\mathcal{F}} w_i \left(n^{\mathrm{UL}}_{i,\max} + n^{\mathrm{DL}}_{i,\max}\right)$ as the new objective function. Then we have the following constraint:

$$z \geq w_i \left(n^{\mathrm{UL}}_{i,\max} + n^{\mathrm{DL}}_{i,\max}\right) \quad (i \in \mathcal{F}). \tag{2.9}$$

By constraint (2.1), at most one of the two terms, $n^{\mathrm{UL}}_{i,\max}$ and $n^{\mathrm{DL}}_{i,\max}$, can be nonzero. Then (2.9) can be reformulated to $z \geq w_i \cdot n^{\mathrm{UL}}_{i,\max}$ and $z \geq w_i \cdot n^{\mathrm{DL}}_{i,\max}$ for $i \in \mathcal{F}$. By definitions of $n^{\mathrm{UL}}_{i,\max}$ and $n^{\mathrm{DL}}_{i,\max}$ in (2.2) and (2.3), we have:

$$n^{\mathrm{UL}}_{i,\max} \geq \sum_{k\in\mathcal{K}} n^{k,\mathrm{UL}}_{(i,j)} \quad (i \in \mathcal{F}, j \in \mathcal{S}_i),$$

$$n^{\mathrm{DL}}_{i,\max} \geq \sum_{k\in\mathcal{K}} n^{k,\mathrm{DL}}_{(i,j)} \quad (i \in \mathcal{F}, j \in \mathcal{S}_i).$$

Therefore, we have the following constraints on z:

$$z \geq w_i \sum_{k\in\mathcal{K}} n^{k,\mathrm{UL}}_{(i,j)} \quad (i \in \mathcal{F}, j \in \mathcal{S}_i), \tag{2.10}$$

$$z \geq w_i \sum_{k\in\mathcal{K}} n^{k,\mathrm{DL}}_{(i,j)} \quad (i \in \mathcal{F}, j \in \mathcal{S}_i). \tag{2.11}$$

The constraints in (2.2), (2.3), (2.4) and (2.5) can be simplified by eliminating $n^{\mathrm{UL}}_{i,\max}$ and

$n^{\mathrm{DL}}_{i,\max}$ and removing the max functions. We have:

$$\sum_{k\in\mathcal{K}} n^{k,\mathrm{UL}}_{(i,j)} \le I^{\mathrm{UL}}_i Q_i \quad (i\in\mathcal{F}, j\in\mathcal{S}_i), \tag{2.12}$$

$$\sum_{k\in\mathcal{K}} n^{k,\mathrm{DL}}_{(i,j)} \le I^{\mathrm{DL}}_i Q_i \quad (i\in\mathcal{F}, j\in\mathcal{S}_i). \tag{2.13}$$

Finally we have the reformulated optimization problem:

OPT-R

minimize z

subject to Adverse impact of LTE on Wi-Fi: (2.10), (2.11),

UL/DL channel assignment: (2.1),

Upper bound on LTE usage: (2.12), (2.13),

Per-user rate requirement: (2.6), (2.7),

$z \ge 0,\ n^{k,\mathrm{UL}}_{(i,j)}, n^{k,\mathrm{DL}}_{(i,j)} \in \mathbb{N},$

$I^{\mathrm{UL}}_i, I^{\mathrm{DL}}_i \in \{0,1\} \quad (i\in\mathcal{F}, j\in\mathcal{S}_i, k\in\mathcal{K}).$

Problem OPT-R is a mixed integer linear program (MILP), one of the most common types of problems for wireless network optimization. Although commercial solvers such as the IBM CPLEX [29] can be employed to compute its optimal solution off-line (useful for benchmark purpose), they cannot meet the stringent real-time scheduling requirement (∼1 ms).

Problem Complexity We have the following result for the computational complexity of our scheduling problem for LTE/Wi-Fi coexistence.

Lemma 2.1. *The scheduling problem for LTE/Wi-Fi coexistence (i.e., minimization of objective (2.8) under the constraints of LTE users' rate requirements and fair LTE/Wi-Fi*

coexistence on each channel) is NP-hard.

Proof. Our proof is based on the set partitioning problem (SPP), which is known to be NP-complete [30]. We consider a decision version of our LTE scheduling problem and construct a polynomial-time reduction from SPP to our problem. We will show that an SPP instance is feasible if and only if the corresponding instance of our constructed problem is feasible, which completes the proof.

SPP is defined as follows. Given a set of positive integers $\mathcal{A} = \{a_1, a_2, \cdots, a_N\}$, determine whether or not $\mathcal{A}$ can be partitioned into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$, where $\mathcal{A}_1 \subset \mathcal{A}$ and $\mathcal{A}_2 = \mathcal{A} \setminus \mathcal{A}_1$, such that the sums of elements in $\mathcal{A}_1$ and $\mathcal{A}_2$ are identical , i.e., $\sum_{a_j \in \mathcal{A}_1} a_j = \sum_{a_j \in \mathcal{A}_2} a_j = \frac{1}{2} \cdot \sum_{a_j \in \mathcal{A}} a_j$.

A decision version of the LTE scheduling problem is to determine whether or not under a given collection of parameters ($R^{k,\mathrm{UL}}$, $R^{k,\mathrm{DL}}$, $C^{k,\mathrm{UL}}_{(i,j)}$, $C^{k,\mathrm{DL}}_{(i,j)}$, Q_i, w_i for all $k \in \mathcal{K}, i \in \mathcal{F}, j \in \mathcal{S}_i$), there exists a feasible TRB-to-user scheduling solution satisfying all constraints with an objective value no more than Z ($Z > 0$).

Consider an arbitrary instance of SPP involving a set $\mathcal{A}$ of N positive integers $a_1, \cdots, a_N$. Solving such an SPP instance is to find a strategy that assigns each element $a_j \in \mathcal{A}$ to either $\mathcal{A}_1$ or $\mathcal{A}_2$ such that $\sum_{a_j \in \mathcal{A}_1} a_j = \sum_{a_j \in \mathcal{A}_2} a_j$. We now construct a corresponding instance of our problem. Our constructed instance consists of two users named user 1 and 2 ($\mathcal{K} = \{1, 2\}$), one channel named channel 1 ($\mathcal{F} = \{1\}$), and N sub-channels on this channel named sub-channel $1, \cdots, N$ ($\mathcal{S}_1 = \{1, \cdots, N\}$). Given the N positive integers $a_1, \cdots, a_N$, input parameters to our problem instance are set as: $N_{\mathcal{SF}} = 1$, $Q_1 = 1$, $w_1 = 1$, $C^{1,\mathrm{UL}}_{(1,j)} = C^{2,\mathrm{UL}}_{(1,j)} = a_j$ and $C^{1,\mathrm{DL}}_{(1,j)} = C^{2,\mathrm{DL}}_{(1,j)} = 0$ for $j \in \{1, \cdots, N\}$, $R^{1,\mathrm{UL}} = R^{2,\mathrm{UL}} = \frac{1}{2} \cdot \sum_{a_j \in \mathcal{A}} a_j$, $R^{1,\mathrm{DL}} = R^{2,\mathrm{DL}} = 0$, and $Z = 1$. We aim to determine whether there exists a feasible scheduling solution for allocating the N TRBs on channel 1 (since $Q_1 = 1$) to user 1 and

2, such that their rate requirements are met and the objective value is no more than $Z = 1$ (the objective value is $+\infty$ if there is no feasible solution). Since $R^{1,\mathrm{DL}} = R^{2,\mathrm{DL}} = 0$, we can readily fix channel assignment variables $I_1^{\mathrm{UL}} = 1$ and $I_1^{\mathrm{DL}} = 0$. Then we have $n_{(1,j)}^{k,\mathrm{DL}} = 0$ for all $k \in \{1, 2\}$ and $j \in \{1, \cdots, N\}$. What remains to be determined is how to allocate TRBs for UL transmission, i.e. fixing variables $n_{(1,j)}^{k,\mathrm{UL}}$ for $k \in \{1, 2\}$ and $j \in \{1, \cdots, N\}$.

The reduction from the SPP instance to our constructed problem instance is as follows. Assigning $a_1, \cdots, a_N$ to $\mathcal{A}_1$ and $\mathcal{A}_2$ corresponds to allocating the N TRBs to user 1 and 2 for UL transmission. Each element a_i is assigned to at most one of the two subsets, which corresponds to our constraint that each TRB can be allocated to at most one of the two users. Specifically, if an element a_j ($j \in \{1, \cdots, N\}$) is assigned to $\mathcal{A}_1$ (or $\mathcal{A}_2$), correspondingly, for our problem we fix $n_{(1,j)}^{1,\mathrm{UL}} = 1$, $n_{(1,j)}^{2,\mathrm{UL}} = 0$ (or $n_{(1,j)}^{1,\mathrm{UL}} = 0$, $n_{(1,j)}^{2,\mathrm{UL}} = 1$). Such reduction also applies reversely from our problem instance to the SPP instance. Clearly, the reduction is on the order of $O(N)$ and is polynomial in time.

We now verify that an SPP instance is feasible (i.e., the set $\mathcal{A}$ can be partitioned into two subsets with equal sum of elements) if and only if our problem instance is feasible (i.e., rate requirements $R^{1,\mathrm{UL}}$ and $R^{2,\mathrm{UL}}$ can be met with the objective equal to $Z = 1$). Indeed, if SPP has a feasible partition strategy satisfying $\sum_{a_j \in \mathcal{A}_1} a_j = \sum_{a_j \in \mathcal{A}_2} a_j = \frac{1}{2} \cdot \sum_{a_j \in \mathcal{A}} a_j$, then based on the reduction we have $\sum_{j=1}^{N} n_{(1,j)}^{1,\mathrm{UL}} C_{(1,j)}^{1,\mathrm{UL}} = \sum_{a_j \in \mathcal{A}_1} a_j = \frac{1}{2} \cdot \sum_{a_j \in \mathcal{A}} a_j = R^{1,\mathrm{UL}}$ and $\sum_{j=1}^{N} n_{(1,j)}^{2,\mathrm{UL}} C_{(1,j)}^{2,\mathrm{UL}} = \sum_{a_j \in \mathcal{A}_2} a_j = \frac{1}{2} \cdot \sum_{a_j \in \mathcal{A}} a_j = R^{2,\mathrm{UL}}$, i.e., our problem instance is feasible. On the other hand, if our problem has a feasible scheduling solution that meets rate requirements $R^{1,\mathrm{UL}}$ and $R^{2,\mathrm{UL}}$, then following the reduction we can determine the partition strategy for the SPP instance that satisfies $\sum_{a_j \in \mathcal{A}_1} a_j = \sum_{a_j \in \mathcal{A}_2} a_j$. This completes the proof.

□

The NP-hardness result suggests that there is no efficient polynomial-time algorithm to solve the problem exactly. In OPT-R, the numbers of variables and constraints are both on the order of $O(|\mathcal{F}||\mathcal{S}_i||\mathcal{K}|)$. For instance, suppose $|\mathcal{F}| = 5$, $|\mathcal{S}_i| = 100$ (20 MHz channel bandwidth), $|\mathcal{K}| = 20$ and $T_{\text{SF}} = 30$ ms. The searching space in OPT-R involves 20011 variables and 22056 constraints. The state-of-the-art solver CPLEX (based on BB-CP) would take 10s of seconds to hours (refer to Section 2.7) to get an optimal or lower-bound solution.

2.5 CURT – A Novel Real-Time Scheduler

Before presenting the design of CURT, we first explain why conventional methods fail to meet our target of providing near-optimal solution to OPT in real-time. We consider the following approaches: (i) reusing LTE schedulers designed for licensed bands, (ii) solving linear programming (LP) relaxation of OPT-R and rounding up the solution to integers, and (iii) using an exact algorithm such as BB to solve OPT-R directly.

Existing LTE schedulers designed for licensed bands are typically metric-based algorithms that allocate TRBs on a per-TTI basis [22]. Specifically, in every TTI, each TRB (or group of TRBs) is allocated to the user that has the highest metric associated with it (e.g., achievable rate, rate requirement, delay, or fairness index). These schedulers, although meeting their real-time requirements, cannot be readily extended to solve problem OPT. This is because OPT has an objective of minimizing LTE's impact on Wi-Fi across multiple channels while meeting LTE users' traffic requirements and constraints. This problem formulation is completely different from those modeled for licensed bands with the objective of maximizing spectral efficiency. Those schedulers have no consideration of the impact of LTE transmission on Wi-Fi and do not address how to optimally divide each CSAT cycle into "on/off"

periods across multiple channels. Thus they cannot be used to solve the scheduling problem in this chapter.

Standard optimization techniques such as LP relaxation and BB cannot meet the real-time requirement of ∼1 ms. Although an LP relaxation of OPT-R can be solved efficiently by Simplex or interior-point methods, the computation time is much larger than 1 ms. BB can be used to solve an MILP such as OPT-R. The basic idea of a BB-based approach is to find upper and lower bounds of each sub-problem as well as the global upper and lower bounds across all sub-problems at each iteration. The gap between upper and lower bounds is expected to shrink after each iteration until it is within the desired optimality gap. The main problem with such an approach is that the overall running time to close the gap is too long to meet our real-time requirement.

2.5.1 An Overview of CURT

Different from existing LTE schedulers used on licensed bands, CURT jointly addresses time division between LTE and Wi-Fi across multiple channels and TRB allocation within LTE "on" periods, with the target of getting optimal or near-optimal solution to OPT-R in real-time (∼1 ms). The design of CURT is based on problem decomposition and parallel execution of sub-problems on a massive number of GPU cores. To pursue near-optimality, CURT first decomposes OPT-R into a large number of sub-problems, each with a fixed assignment of UL/DL channels and "off/on" time division pattern across all channels, and then solves all sub-problems in parallel by GPU cores. In particular, our proposed decomposition ensures that there is no inter-dependency among sub-problems, so that all sub-problems can be executed independetly and in parallel. Thus the time complexity of solving OPT-R is reduced to that of solving one sub-problem. Further, since all sub-problems are of the same structure,

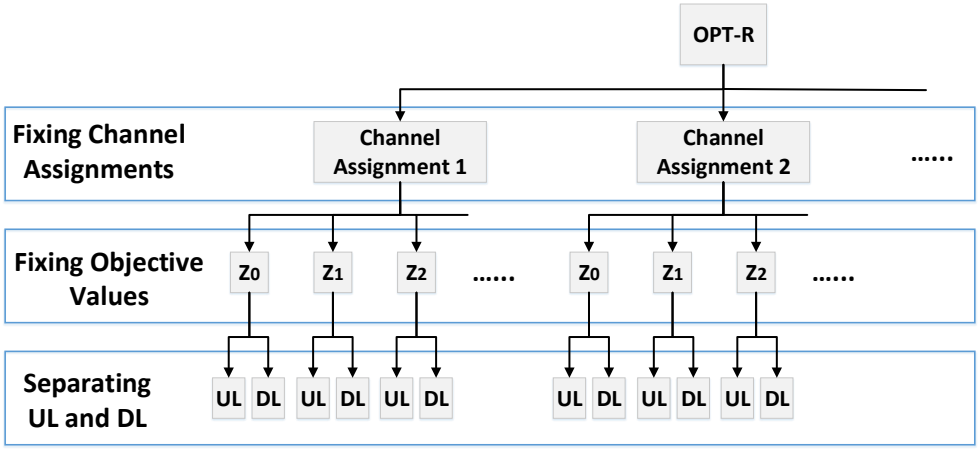


Figure 2.4: Decomposition of OPT-R.

they can be solved within a close-to-same amount of time. The parallel design of CURT ensures that all possible cases of LTE's impact on Wi-Fi can be evaluated (for feasibility) via a simple and fast algorithm in parallel. By comparing among the objectives of all feasible solutions, CURT has a high probability to find a near-optimal solution.

2.5.2 Problem Decomposition

Figure 2.4 shows how we decompose OPT-R into small independent sub-problems. There are three levels of decomposition: 1) fixing UL and DL channel assignments, 2) fixing achievable objective values, and 3) separating UL and DL sub-problems. Details of these steps are as follows.

Fixing UL/DL Channel Assignments Given a set of channels $\mathcal{F}$ for LTE/Wi-Fi coexistence, there is a total of $\binom{|\mathcal{F}|}{1} + \binom{|\mathcal{F}|}{2} + \cdots + \binom{|\mathcal{F}|}{|\mathcal{F}|-1} = 2^{|\mathcal{F}|} - 2$ different ways for assigning the channels for UL and DL transmissions. In practice, $|\mathcal{F}|$ is not a large number due to limitations on signal processing capability and energy consumption for LTE small cell eNBs and user equipments. For example, when $|\mathcal{F}| = 5$, there is a total of 30 different

channel assignments. Problem OPT-R can thus be decomposed into a set of $(2^{|\mathcal{F}|}-2)$ sub-problems, each with a given UL/DL channel assignment. For each sub-problem, we need to find the smallest objective value that is achievable under the given channel assignment.

Fixing Achievable Objective Values From constraints (2.10) and (2.11), it is clear that the objective value of OPT-R under a given UL/DL channel assignment only has a finite number of possibilities. Specifically, since both $w_i \sum_{k\in\mathcal{K}} n_{(i,j)}^{k,\mathrm{UL}}$ and $w_i \sum_{k\in\mathcal{K}} n_{(i,j)}^{k,\mathrm{DL}}$ (for all $j \in \mathcal{S}_i$ on channel $i \in \mathcal{F}$) can only take values from $\mathcal{Z}_i = \{0, w_i, 2w_i, \cdots, Q_i w_i\}$, the optimal objective value z^* must be within the set

$$\mathcal{Z} = \bigcup_{i\in\mathcal{F}} \mathcal{Z}_i. \tag{2.14}$$

Since each set $\mathcal{Z}_i$ $(i \in \mathcal{F})$ consists of 0 and Q_i nonzero elements (if $Q_i > 0$), we have

$$|\mathcal{Z}| \le 1 + \sum_{i\in\mathcal{F}} Q_i \le |\mathcal{F}| \cdot N_{\mathrm{SF}}, \tag{2.15}$$

where the second inequality follows the definition $Q_i < N_{\mathrm{SF}}$ for all $i \in \mathcal{F}$.

By fixing objective value, we further decompose each sub-problem under a given channel assignment into $|\mathcal{Z}|$ smaller sub-problems. For each resulting sub-problem, we need to determine whether or not a given objective value in $\mathcal{Z}$ is achievable (i.e. feasibility). After this decomposition, the $|\mathcal{Z}|$ sub-problems under a given channel assignment characterize all possible "off/on" time division patterns across channels in $\mathcal{F}$.

Separating UL/DL Sub-Problems For each sub-problem under a given channel assignment and objective value, we need to check whether or not it is feasible to meet all users' UL and DL rate requirements. This feasibility check can again be decomposed into two parallel problems, one for UL and the other for DL.

Now the original problem OPT-R is decomposed into a total of $2(2^{|\mathcal{F}|}-2)\cdot|\mathcal{Z}|$ UL/DL feasibility check sub-problems (each with a given channel assignment and objective value). We propose to employ low-cost off-the-shelf GPU (each consisting of a massive number of processing cores) to solve them in parallel. Once feasibility checks for all sub-problems are completed, we pick the smallest feasible objective value under all UL/DL channel assignments that has both its UL and DL sub-problems pass feasibility checks (both are feasible). The scheduling solution corresponding to this objective value and channel assignment is our output solution.

2.5.3 Feasibility Check of Sub-Problems

In the feasibility check of a UL/DL sub-problem, we aim to determine whether or not each user's UL/DL rate requirement can be met under the given UL/DL channel assignment. This problem can be formulated as an integer linear program (ILP), which is NP-hard and cannot be solved exactly under our tight time constraint ($\sim$ 1 ms). So a fast and efficient heuristic algorithm is needed. In this section, we present the design for feasibility check of DL sub-problems. The case for UL sub-problems is similar and is omitted to conserve space.

For each DL channel i, there are $|\mathcal{S}_i|$ sub-channels on it and we consider one sub-channel at a time to fill users' rate requirements. The order for selecting channels and sub-channels is arbitrary. For a given sub-channel, we use it to fill one or more user's rate requirement. The question is: Which user (among the users whose rate requirements have not been met) should we consider? This is a user selection problem. Note that the sub-channel capacity of each user differs. That is, one user may find the sub-channel to be of good condition while another user may find it otherwise. So first we need to find each user's sub-channel capacity.

The next question is: should sub-channel capacity be the *only* criterion in user selection?

Algorithm 1 Feasibility Check of DL Sub-problem

Input the set of DL channels $\mathcal{F}^{\mathrm{DL}}$ and the objective value z;
Initialize:
    1) $V_{\mathrm{res}}^{k,\mathrm{DL}} := R^{k,\mathrm{DL}} T_{\mathrm{SF}}$ for each $k \in \mathcal{K}$;
    2) Q'_i as in (2.16) for each DL channel $i \in \mathcal{F}^{\mathrm{DL}}$;
    3) Feasibility := False;
while ($\mathcal{F}^{\mathrm{DL}} \neq \emptyset$ **and** Feasibility = False) **do**
    Choose any remaining channel in $\mathcal{F}^{\mathrm{DL}}$ and denote it as channel i;
    while ($\mathcal{S}_i \neq \emptyset$ **and** Feasibility = False) **do**
        Choose any remaining sub-channel in $\mathcal{S}_i$ and denote it as (i,j);
        $Q'_{(i,j)} := Q'_i$;
        while ($Q'_{(i,j)} > 0$ **and** Feasibility = False) **do**
            Find the user $k' := \arg\max_{k \in \mathcal{K}} C_{(i,j)}^{k,\mathrm{DL}} \cdot V_{\mathrm{res}}^{k,\mathrm{DL}}$;
            Set $n_{(i,j)}^{k',\mathrm{DL}} := \min\left\{Q'_i, \left\lceil \frac{V_{\mathrm{res}}^{k',\mathrm{DL}}}{C_{(i,j)}^{k',\mathrm{DL}} T_0} \right\rceil\right\}$;
            Update $V_{\mathrm{res}}^{k',\mathrm{DL}} := V_{\mathrm{res}}^{k',\mathrm{DL}} - n_{(i,j)}^{k',\mathrm{DL}} C_{(i,j)}^{k',\mathrm{DL}} T_0$
            and $Q'_{(i,j)} := Q'_{(i,j)} - n_{(i,j)}^{k',\mathrm{DL}}$;
            if ($V_{\mathrm{res}}^{k',\mathrm{DL}} \leq 0$) **then**
                $\mathcal{K} := \mathcal{K} \setminus \{k'\}$;
            if ($\mathcal{K} = \emptyset$) **then**
                Feasibility := True;
        end while
        $\mathcal{S}_i := \mathcal{S}_i \setminus \{(i,j)\}$;
    end while
    $\mathcal{F}^{\mathrm{DL}} := \mathcal{F}^{\mathrm{DL}} \setminus \{i\}$;
end while
return Feasibility;

The answer is *No.* This is because a user experiencing low capacity on this sub-channel may experience a even lower capacity on other sub-channels. If we do not consider this user on the given sub-channel, it will consume even more TRBs on other sub-channels.

Taking the above two considerations together, we propose a user selection criterion that selects one user (among the remaining users whose rate requirements have not been met) who has the largest sub-channel capacity weighted by its remaining work (rate to be filled).

Our proposed feasibility check algorithm for the DL sub-problem is given in Algorithm 1. The input is a given set $\mathcal{F}^{\mathrm{DL}}$ of DL channels and an objective value z. For convenience, we introduce a new notation $V_{\mathrm{res}}^{k,\mathrm{DL}}$, $k \in \mathcal{K}$, which represents the remaining DL data (in bits)

that should be scheduled for user k within a SF to meet its DL rate requirement. Initially, we have $V_{\text{res}}^{k,\text{DL}} = R^{k,\text{DL}} T_{\text{SF}}$.

Denote Q_i' as the number of TRBs that are available to LTE on each sub-channel $(i,j) \in \mathcal{S}_i$ under the objective value z. Then Q_i' is upper bounded by Q_i. Further, for given z, by (2.11), Q_i' is also upper bounded by $\lfloor z/w_i \rfloor$. We have:

$$Q_i' = \begin{cases} \min\{Q_i, \lfloor \frac{z}{w_i} \rfloor\}, & \text{for } w_i > 0, \\ Q_i, & \text{for } w_i = 0. \end{cases} \tag{2.16}$$

The main body of Algorithm 1 consists of three "while" loops, with the two outer while loops enumerating all remaining DL channels and sub-channels and the most inner while loop filling in users' rate requirements. Specifically, on sub-channel (i,j), we select user $k' \in \mathcal{K}$ (where $\mathcal{K}$ is the set of remaining users) based on the criterion that we discussed earlier, i.e., with the largest $C_{(i,j)}^{k',\text{DL}} \cdot V_{\text{res}}^{k',\text{DL}}$. The number of TRBs allocated to user k' is

$$n_{(i,j)}^{k',\text{DL}} = \min \left\{ Q_i', \left\lceil \frac{V_{\text{res}}^{k',\text{DL}}}{C_{(i,j)}^{k',\text{DL}} T_0} \right\rceil \right\}. \tag{2.17}$$

Then we update the remaining bit volume for this user:

$$V_{\text{res}}^{k',\text{DL}} = V_{\text{res}}^{k',\text{DL}} - n_{(i,j)}^{k',\text{DL}} C_{(i,j)}^{k',\text{DL}} T_0. \tag{2.18}$$

When $V_{\text{res}}^{k',\text{DL}} \leq 0$, i.e., the user's rate requirement is met, we remove this user from $\mathcal{K}$. If there are still remaining TRBs on this sub-channel (i,j), we continue to select another user from $\mathcal{K}$ (with the same criterion) and follow the same TRB allocation process. Once all TRBs on this sub-channel are allocated, we move on to the next sub-channel and eventually the next channel.

Algorithm 1 terminates when either all users' DL rate requirements are met (i.e., $\mathcal{K} = \emptyset$) or TRBs on all DL channels (and sub-channels) are already allocated (i.e., $\mathcal{F}^{\mathrm{DL}} = \emptyset$). The DL sub-problem is infeasible if there remains some user with $V_{\mathrm{res}}^{k,\mathrm{DL}} > 0$ after all TRBs are allocated. Otherwise, we conclude that it is feasible.

2.5.4 Computational Complexity

The time complexity of CURT is determined by the feasibility check of a DL/UL sub-problem, while its space complexity depends on the number of DL/UL sub-problems.

For each DL/UL feasibility check, we need to go through at most $|\mathcal{F}||\mathcal{S}_i|$ sub-channels. On each sub-channel, user selection is on the order of $O(|K|)$. So the time complexity of a feasibility check is $O(|\mathcal{F}||\mathcal{S}_i||\mathcal{K}|)$.

For space complexity, we need to determine how many processors are needed for solving all feasibility checks in parallel. From our analysis in Section 2.5.2, we know that the total number of UL/DL sub-problems is $2(2^{|\mathcal{F}|} - 2)|\mathcal{Z}|$. Based on (2.15), it is upper bounded by $2(2^{|\mathcal{F}|} - 2)|\mathcal{F}|N_{\mathrm{SF}}$, which is independent of the number of LTE and Wi-Fi users. That is, the number of GPU cores required by CURT does not increase with the number of LTE or Wi-Fi users.

2.5.5 Guaranteeing Feasibility via Traffic Management

Problem OPT/OPT-R may have no feasible solution to meet the constraints on rate requirements for all $k \in \mathcal{K}$ and LTE's channel usage Q_i's for all $i \in \mathcal{F}$. On the other hand, CURT may not be able to find a feasible solution when OPT-R is actually feasible since CURT does not traverse the entire search space of OPT-R. In fact, it is impossible for any algorithm

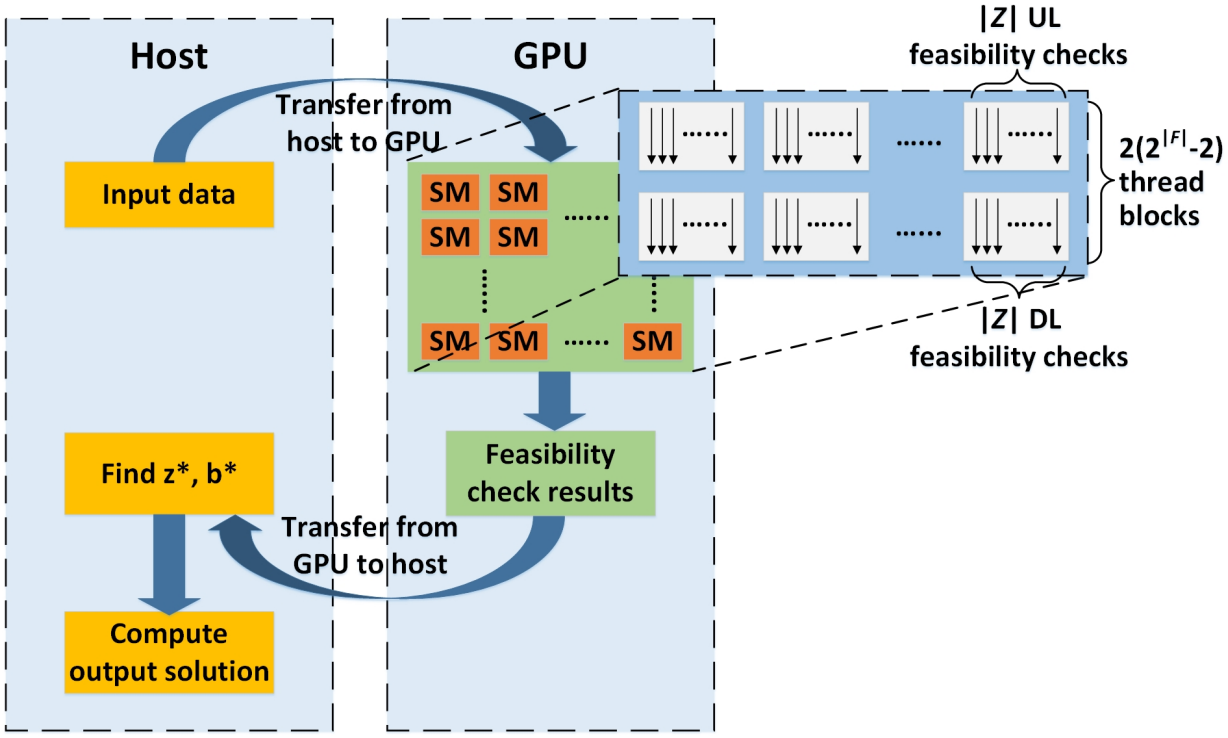


Figure 2.5: An illustration of our implementation of CURT.

to go through all possible solutions of an NP-hard problem (such as OPT-R) while meeting the real-time requirement of ∼1 ms. To guarantee feasibility, CURT should work in concert with a traffic management mechanism. Specifically, if a user's UL/DL rate requirements cannot be fully met after scheduling a SF, the traffic management module should negotiate with this user to decrease its rate requirements or switch it to a band on licensed spectrum. Detailed design of such a traffic management module is beyond the scope of this chapter.

2.6 Implementation

As a proof of concept, we implement CURT using off-the-shelf NVIDIA Quadro P6000 GPUs [28] based on the CUDA programming model [25]. Our implementation is done on a Dell desktop computer with an Intel Xeon E5-2687W v4 CPU (3.0 GHz) and dual

NVIDIA Quadro P6000 GPUs. Each Quadro P6000 GPU consists of an array of 30 streaming multiprocessors (SMs) with 3840 CUDA cores (128 cores per SM). In each SM, there is 96 KB shared memory.[3] CUDA is a programming model for general-purpose parallel computing on NVIDIA GPUs. Logically, CUDA executes a multi-threaded function (termed a kernel) on GPU through a hierarchy of threads. This hierarchy has a two-layer structure, where the upper layer is a grid of thread blocks, and at the lower layer each block consists of a number of threads. Each block is executed by a single SM and an SM addresses one block at a time. All blocks are queued and scheduled among the available SMs on the GPU. A thread is the smallest computing granularity under CUDA. For current GPUs, the maximum number of threads allowed per thread block is 1024. Threads within a block are handled by CUDA cores on the assigned SM and can communicate among each other via shared memory.

Fig. 2.5 illustrates our implementation, which consists of four stages: (i) transferring input data from host (CPU) memory to GPU global memory; (ii) performing parallel UL/DL feasibility checks in GPU (refer to Section 2.5.3); (iii) transferring results of feasibility checks from GPU back to host; and (iv) determining the output scheduling solution on the host. Next we present details of these four stages.

Transferring Input Data to GPU Input data to CURT are classified into *fast-varying* and *slow-varying* data, depending on how fast they vary in time. A classification of input data is given in Table 2.2. Fast-varying data refer to those that vary from SF to SF and thus must be uploaded to GPU for each SF. In each SF, we transfer fast-varying data from host to GPU before executing the kernel for feasibility checks. On the other hand, slow-varying data does not vary from SF to SF and only need to be updated over a time period much longer than a SF (hundreds of ms or more). As these data do not change per SF, we define

[3]Shared memory is on-chip and locates at each SM, which can only be accessed by cores within an SM. In contrast, GPU's global memory is off-chip and accessible to cores from all SMs. Access to shared memory is much faster than access to GPU's global memory [26].

Table 2.2: Classification of input to CURT.

Data	**Time Variation**	**Transferring to GPU per SF**
T_{SF} (also N_{SF})	Slow	No
$\mathcal{F}$		
$\mathcal{S}_i$ for $i \in \mathcal{F}$		
$\mathcal{K}$		
w_i for $i \in \mathcal{F}$		
Q_i for $i \in \mathcal{F}$		
$R^{k,\mathrm{UL}}, R^{k,\mathrm{DL}}$ for $k \in \mathcal{K}$		
The $(2^{\lvert\mathcal{F}\rvert} - 2)$ possible UL/DL channel assignment patterns		
The set $\mathcal{Z}$ of possible objective values of OPT-R		
$C^{k,\mathrm{UL}}_{(i,j)}, C^{k,\mathrm{DL}}_{(i,j)}$ for $i \in \mathcal{F}, j \in \mathcal{S}_i, k \in \mathcal{K}$	Fast	Yes

a separate kernel to transfer them from host to GPU only when they vary.

When measuring the scheduling time of CURT, we include the transferring time for fast-varying data since they must be updated to GPU per SF. The time cost for updating slow-varying data is not incorporated in the scheduling time as such transfer only occurs on a much larger time scale.

Performing Feasibility Checks on GPU When all input data is available in GPU's global memory, the second stage is to perform feasibility check on a total of $2(2^{|\mathcal{F}|} - 2) \cdot |\mathcal{Z}|$ UL/DL sub-problems (refer to Section 2.5.2) in parallel by a kernel with a grid of thread blocks. We need to take the following factors into consideration when designing the kernel: 1) the number of available SMs; 2) the capacity of shared memory on each SM; and 3) the design of feasibility checks. First, since an SM can only execute one thread block at a time and works sequentially if it is assigned with multiple blocks, the total number of thread blocks in the grid should match the number of SMs to maximize occupancy while minimizing

sequential operations on SMs. Next, to meet the time requirement of ∼1 ms, we should make the best use of shared memory on each SM and reduce access to global memory. However, the size of all input data of CURT (see Table 2.2) may exceed the capacity of shared memory per SM (96 KB for NVIDIA Quadro P6000 GPU). We need to ensure that the shared memory used by each thread block is within the per-SM capacity limit. Last, our proposed feasibility check algorithm (Algorithm 1) is of sequential design and would be used on a large number of independent UL/DL sub-problems. To maximize parallelism, we should properly distribute all feasibility checks among the defined thread blocks and utilize the large number of threads per block for concurrent processing.

With the above considerations, we use a kernel with a one-dimensional grid of $2(2^{|\mathcal{F}|}-2)$ thread blocks for this stage. Each thread block addresses the $|\mathcal{Z}|$ feasibility checks for all UL (or all DL) sub-problems under one of the $(2^{|\mathcal{F}|}-2)$ UL/DL channel assignments. Detailed operations are:

- *Step 1:* At the beginning of the kernel, we use all 1024 threads in each block to load input data (for either UL or DL) from global memory into shared memory successively in a round-by-round manner (1024 elements per round).
- *Step 2:* After loading input data, we proceed to run the $|\mathcal{Z}|$ feasibility checks in each block. Each feasibility check is executed by a single thread. We only keep result of the check, i.e., feasible or infeasible, while the scheduling solution is discarded (not stored in either shared memory or global memory). More explanation on this will be given later in the next stage.
- *Step 3:* When feasibility checks on all thread blocks are completed, we store the check results into GPU's global memory.

The above kernel structure meets all of our design considerations. Since $|\mathcal{F}|$ is not a large

number and the number of thread blocks would be comparable to that of available SMs. For example, for $|\mathcal{F}| = 5$ [19], we have $2(2^{|\mathcal{F}|}-2) = 60$, which is equal to the number of SMs from dual Quadro P6000 GPUs that we use for implementation. In addition, since each thread block only addresses sub-problems for one transmission direction (UL or DL), we only need to load input data of the specific direction into the shared memory of the assigned SM. Thus the required shared memory for input data per thread block (SM) is reduced by half. Further, feasibility checks on all thread blocks run concurrently and in parallel, with no need for communication among blocks and threads. From (2.15), the number of parallel feasibility checks that each thread block needs to execute is upper bounded by $|\mathcal{Z}| \leq |\mathcal{F}| \cdot N_{\text{SF}}$, which does not exceed the maximum number of threads per block, i.e., 1024. For example, for $|\mathcal{F}| = 5$ and $N_{\text{SF}} = 30$ (refer to Section 2.3), we have $|\mathcal{Z}| \leq 150$.

Transferring Feasibility Results to Host When the GPU kernel completes all feasibility checks and stores these results in GPU's global memory, we transfer these results back to the host memory. This transferring time overhead is included in the total scheduling time.

Determining Output Scheduling Solution on Host The feasibility check results obtained from GPU indicate whether or not each UL/DL sub-problem under a specific channel assignment and objective value is feasible. We now determine the final output scheduling solution on the host through the following operations:

- *Step 1:* First, we find the smallest objective value $z^* \in \mathcal{Z}$ under the best channel assignment b^* (among the $(2^{|\mathcal{F}|} - 2)$ assignments) that has both the corresponding UL and DL sub-problems pass feasibility checks (both are feasible). This is done on the host by traversing the feasibility check results. Specifically, under each channel assignment b, we have one UL sub-problem and one DL sub-problem for each objective value $z \in \mathcal{Z}$. Denote z_b^* as the smallest objective value (if exists) under channel assignment b with its both UL and DL sub-problems being feasible. Then we have

$z^* = \min_b z_b^*$ and $b^* = \arg\min_b z_b^*$.

- *Step 2:* We then run Algorithm 1 on the host with the input of objective value z^* and channel assignment b^*. The obtained scheduling solution is the final output solution of CURT.

The computational time of this stage is included in the total scheduling time of CURT.

The reason why we do not store candidate solutions on GPU and instead re-compute the output solution on the host is as follows. First, storing candidate solutions on GPU during feasibility checks would result in a large amount of access to GPU's global memory (as shared memory is insufficient). But this is unacceptable under our tight time constraint of $\sim$1 ms. Second, transferring solutions from GPU to host is actually more time-consuming than doing a sequential check of results and re-computing the scheduling solution (using Algorithm 1) one more time on the host.

2.7 Validation

In this section, we validate and evaluate the performance of CURT through experiments.

2.7.1 Experimental Platform and Network Parameters

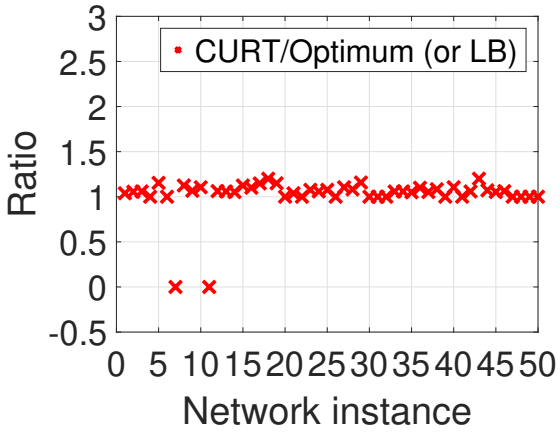
Our experiments are done on a Dell desktop computer with an Intel Xeon E5-2687W v4 CPU (3.0 GHz) and dual NVIDIA Quadro P6000 GPUs (each with 30 SMs and 3840 CUDA cores). The communication bus between CPU and GPU is a PCIe 3.0 X16 slot with default configuration. Implementation on GPU is based on the NVIDIA CUDA version 9.2 programming model [25]. We employ IBM CPLEX Optimizer version 12.7.1 [29] on the same

computer to compute optimal or lower-bound solution to OPT-R.[4] During our experiments, one of the two GPUs (GPU 1) also performs graphics processing and display functions for the desktop computer, in addition to the computational task associated with CURT, while the other GPU (GPU 2) is solely dedicated to the computational task associated with CURT. Since CURT would be implemented on a small cell eNB and does not need to perform graphics processing and display as in a desktop computer, it is more reasonable to use the timing results from GPU 2 to demonstrate CURT's performance.

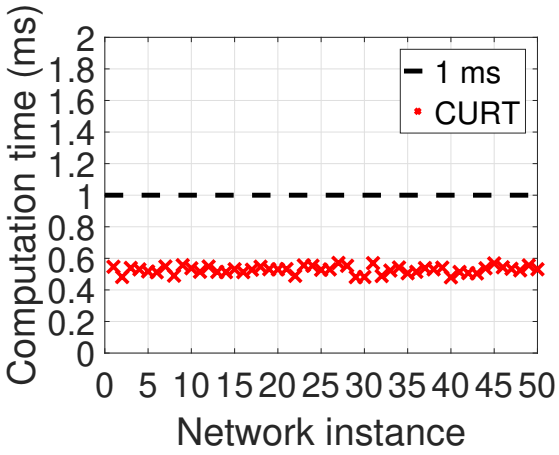
We assume that all LTE and Wi-Fi users in the small cell are within each other's transmission and interference ranges and there is no hidden-node. Suppose that $|\mathcal{F}| = 5$ channels in the 5 GHz unlicensed spectrum are chosen for LTE/Wi-Fi coexistence, with carrier frequencies being 5.20, 5.22, 5.24, 5.26, and 5.28 GHz, respectively. Each channel has 20 MHz bandwidth and is divided into $|\mathcal{S}_i| = 100$ sub-channels (each with 180 kHz bandwidth). The time duration of a SF $T_{\text{SF}} = 30$ ms. To facilitate reproducibility of the results, we use Shannon's formula to calculate data rates $C^{k,\text{UL}}_{(i,j)}$'s and $C^{k,\text{DL}}_{(i,j)}$'s, i.e., $C^{k,\text{UL}}_{(i,j)} = B\log_2\left(1+\frac{\rho^{\text{UL}} l^k_i |h^k_{(i,j)}|^2}{\sigma_0^2}\right)$ and $C^{k,\text{DL}}_{(i,j)} = B\log_2\left(1+\frac{\rho^{\text{DL}} l^k_i |h^k_{(i,j)}|^2}{\sigma_0^2}\right)$, where B is the bandwidth of a sub-channel, ρ^{UL} and ρ^{DL} are the UL and DL transmit power of an LTE user on each sub-channel, respectively, l^k_i denotes the path-loss between user k and the eNB on channel i, $h^k_{(i,j)}$ denotes the Rayleigh fading coefficient between user k and the eNB on sub-channel (i,j) with mean 0 and variance 1, and σ_0^2 is the noise power. The pathloss is modeled by the Friis transmission equation $l^k_i = G_t G_r \left(\frac{\lambda_i}{4\pi D_k}\right)^2$, where G_t and G_r are respectively the transmit and receive antenna gains, λ_i is the wavelength on channel i, and D_k denotes the distance between the eNB and user k.

Antenna gains are set to $G_t = G_r = 1$. Since transmit power of eNB is typically higher

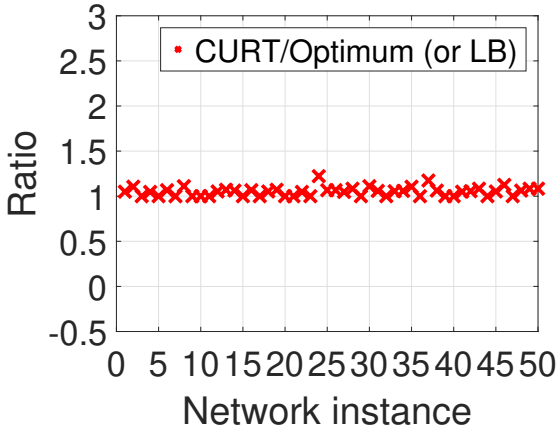
[4]To address potentially prohibitively long computation time by CPLEX, we set a time limit of 1 hour. The lower-bound solution is taken as benchmark when CPLEX cannot find optimal solution by 1 hour.



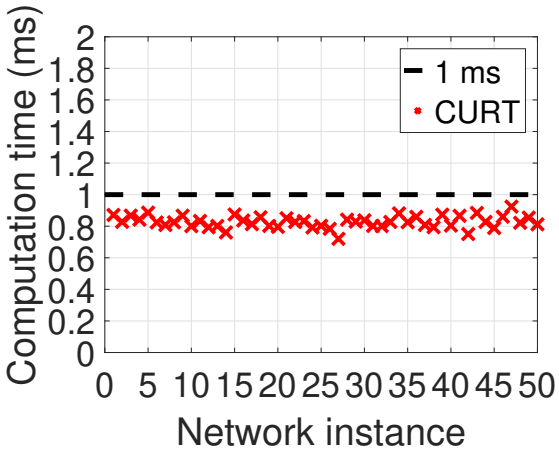
(a) Ratios of objectives. $|\mathcal{K}| = 10$.



(b) Computation time. $|\mathcal{K}| = 10$.



(c) Ratios of objectives. $|\mathcal{K}| = 20$.



(d) Computation time. $|\mathcal{K}| = 20$.

Figure 2.6: Achieved objective value and timing performance of CURT.

than that of user terminals, we set $\rho^{\mathrm{DL}}/\sigma_0^2 = 120$ dB and $\rho^{\mathrm{UL}}/\sigma_0^2 = 115$ dB. For Q_i, we let $Q_i = \left\lfloor N_{\mathrm{SF}} \frac{|\mathcal{K}|/|\mathcal{F}|}{|\mathcal{K}|/|\mathcal{F}|+U_i} \right\rfloor$ as described in Section 2.4.

2.7.2 Results

Optimality and Timing Performance We now evaluate the performance of CURT under realistic network settings. Following 3GPP's evaluation methodology [5], we consider a maximum of 20 users in a LTE small cell. In Fig. 2.6, we show results for both $|\mathcal{K}| = 10$ and $|\mathcal{K}| = 20$ users. In both cases, distances between eNB and LTE users are randomly and uniformly generated from $[1, 30]$ m. As we discussed in Section 2.5.4, the computational complexity of CURT is independent of the number of Wi-Fi users. Without loss of generality, we assume that U_i on each channel $i \in \mathcal{F}$ is randomly chosen from $\{1, 2, 3\}$ so that on average there are $\sim$10 Wi-Fi users sharing the spectrum with LTE, which is similar to the evaluation scenario in [5].

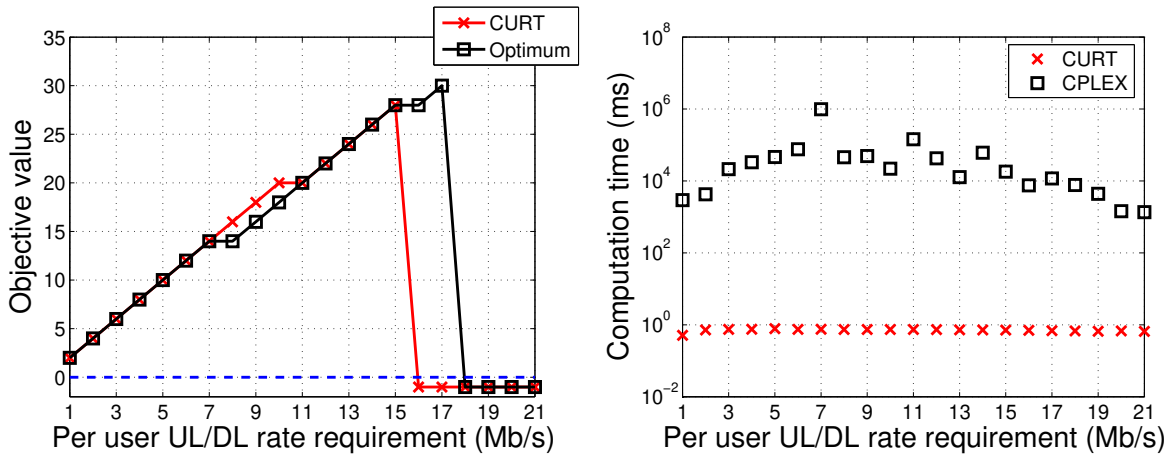
In Fig. 2.6(a) and (b), UL and DL rate requirements of the 10 users are randomly generated from $[10, 40]$ Mb/s. Fig. 2.6(a) shows ratios between objective values achieved by CURT and optimal (or lower-bound) solutions found by CPLEX over 50 network instances. For each instance, if CURT finds the optimum, the ratio of objective values equals to one. When CURT fails to find a feasible solution, we set the ratio to zero. Among 50 network instances, CURT finds optimal solutions for 14 instances. We observe that in 2 instances CURT cannot find feasible solution.[5] That is, the percentage that CURT can find a feasible solution is 96%. This is reasonable since CURT does not traverse the entire search space of OPT-R. Among the instances where CURT can find feasible solutions, the average of CURT's ratios is 1.04, with a variance of 0.0021.

[5] In this case, the traffic management module may be invoked to ensure feasibility.

Fig. 2.6(b) shows computation time of CURT. Mean computation time of CURT is 0.53 ms, with a maximum of 0.57 ms and a variance of 0.0006. In contrast, the mean computation time of CPLEX for finding the optimal (or lower-bound) solution is 1246.35 s.

In Fig. 2.6(c) and 2.6(d), the rate requirements of the 20 users are randomly generated from $[5, 20]$ Mb/s. Fig. 2.6(c) shows that for 18 out of 50 instances, CURT achieves optimal objectives. Also, for all 50 instances, CURT is able to find feasible solutions. The average of ratios by CURT (over optimum) is 1.04, with a variance of 0.0014. Mean computation time of CURT is 0.83 ms, with a maximum of 0.92 ms and a variance of 0.0014. In contrast, the mean of CPLEX's computation time for finding optimal or lower-bound solution is 987.86 s. Numerical results in Fig. 2.6 demonstrate that CURT can indeed find near-optimal solution while meeting the real-time requirement of $\sim$1 ms.

Note that the computational time of CPLEX with 10 users is higher than that with 20 users. This is because the computational time of CPLEX depends on a number of factors, with the number of users in the cell being just one factor. CPLEX is based on branch-and-bound algorithm with cutting plane method, which closes the optimality gap by iteratively comparing the difference between the objective of the best feasible solution and the lower-bound (or upper-bound). In general, its computational time increases with the number of decision variables. But the amount of time in each iteration depends heavily on the local search algorithm to find a feasible solution. For problem OPT, although there are fewer variables in the case of 10 users, it is harder to find a feasible solution as the rate requirement for each user ranges from 10 to 40 MB/s. On the other hand, when there are 20 users, the rate requirement for each user ranges from 5 to 20 MB/s (a narrower range). In this case, the time to find a feasible solution is actually smaller (than 10 users). In contrast, CURT does not follow branch-and-bound framework. Its computational time does not involve a local search to find a feasible solution. Its time complexity (for executing parallel feasibility

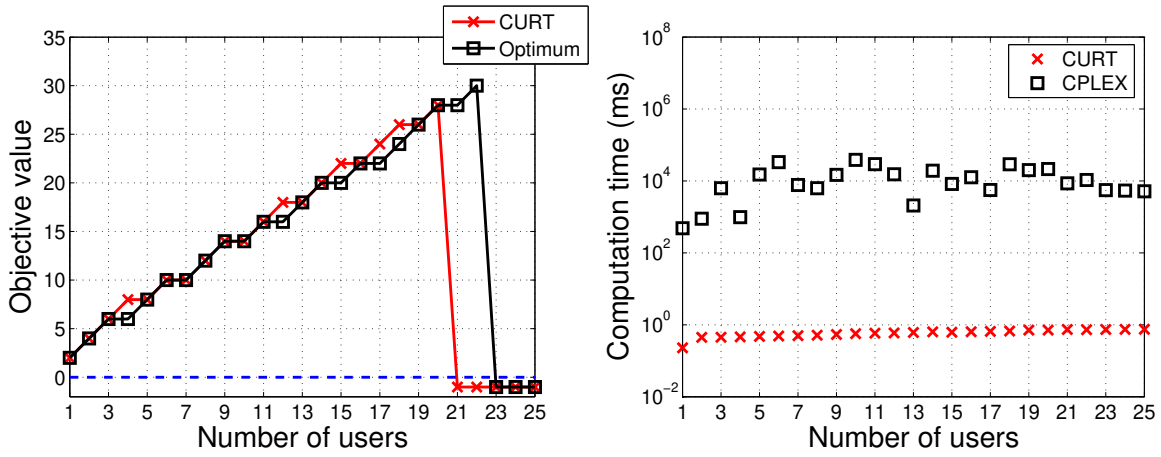


(a) Objective value. (b) Computation time.

Figure 2.7: Performance of CURT under increasing per-user rate requirement. Objective value of -1 indicates infeasibility.

check in each sub-problem) grows linearly with the number of users (refer to Section 2.5.4).

Varying LTE Traffic Load We now evaluate the behavior of CURT under varying LTE traffic load. Suppose the cell has 30 users. To identify each user distinctly, we name them as user 1 to 30. A user's channel capacity is a function of its distance to the base station, in addition to time-varying channel fading. The number of TRBs required in scheduling depends on each user's channel capacity and its rate requirement. With a given set of user rate requirements, the higher the users' channel capacity, the fewer TRBs need to be allocated and more users can be accommodated. Thus, for evaluating the number of users that can be supported by CURT, it is necessary to specify users' distances to the eNB. For reproducibility, we hereby disclose users' distances (all randomly generated) to the eNB as follows (in meter): 24.58, 1.29, 5.03, 6.88, 6.76, 18.51, 8.89, 6.77, 1.44, 22.66, 13.91, 28.02, 14.51, 13.14, 25.54, 16.23, 6.88, 20.49, 25.31, 1.57, 20.76, 12.01, 25.12, 15.58, 21.57, 13.44, 9.83, 6.50, 6.61, 20.78. Note that the LTE cell may not be able to meet rate requirements of all these users. It may only serve a subset of users and transfer the remaining users to licensed band (via the traffic management module) as described in Section 2.5.5. We set



(a) Objective value. (b) Computation time.

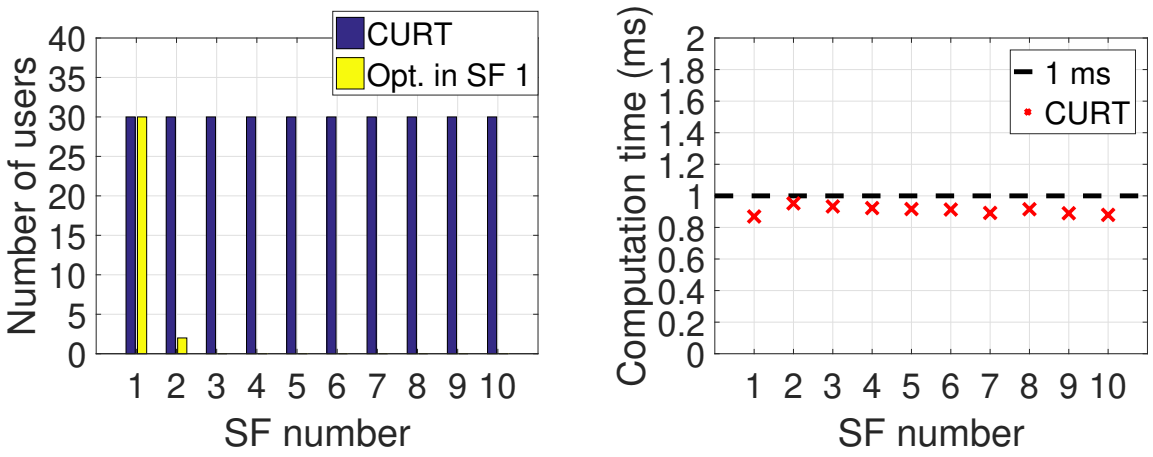
Figure 2.8: Performance of CURT under increasing number of users. Objective value of -1 indicates infeasibility.

$Q_i = N_{\text{SF}}/2$ and $U_i = 2$ for all channels in $\mathcal{F}$.

In Fig. 2.7(a), we choose the first 20 users and increase their UL/DL rate requirements simultaneously from 1 Mb/s. We see that CURT can support a maximum per-user UL/DL rate requirement of 15 Mb/s, while the optimal solution can support up to 17 Mb/s. On the other hand, in Fig. 2.7(b), we see that CURT's computation time is consistently less than 1 ms while CPLEX's computation time varies from 1.36 s to 984.33 s.

In Fig. 2.8(a), we fix the per-user UL/DL rate requirements to 15 Mb/s and increase the number of LTE users (starting from user 1). It shows that CURT can satisfy the first 20 users, while the optimal solution can support the first 22 users. In Fig. 2.8(b) we see that computation time of CURT is no more than 1 ms while CPLEX's computation time varies from 484 ms to 38.47 s.

Temporal Channel Variations In practical LTE small cells, generally user mobility is low and channel conditions may only change slightly across consecutive SFs. We now show that it is still necessary to re-compute scheduling solution for each SF. We employ a quasi-



(a) Number of users meeting rate requirements.

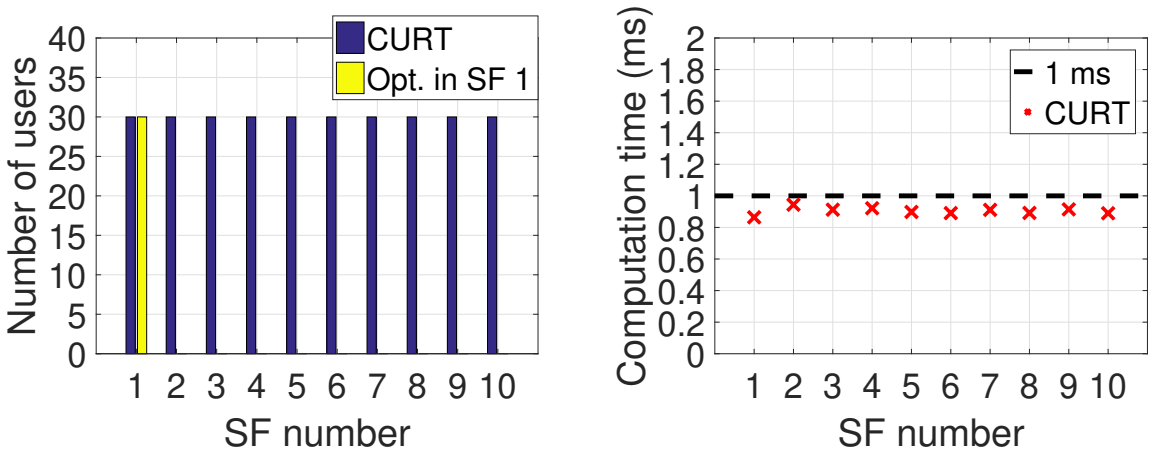
(b) Computation time of CURT.

Figure 2.9: Performance of CURT under temporal channel variations for 30 users. User mobility $v = 1.5$ km/h.

static block fading channel model to incorporate temporal channel correlations [27]. Assume that fast fading coefficients $h^k_{(i,j)}$'s remain constant during an SF and vary in the next SF. Denote $h^k_{(i,j)}(t)$ as the fast fading coefficient of user k on sub-channel (i,j) in SF t. Then the fast fading coefficient in SF $(t+1)$ is determined by $h^k_{(i,j)}(t+1) = \alpha h^k_{(i,j)}(t) + \tilde{h}^k_{(i,j)}$, where α represents the temporal autocorrelation between consecutive SFs, and $\tilde{h}^k_{(i,j)}$ is the uncorrelated Rayleigh channel variation with mean 0 and variance $(1-\alpha^2)$. α is calculated as $\alpha = J_0(2\pi f_{\mathrm{M}} T_{\mathrm{SF}})$ [27], where $J_0(\cdot)$ is zeroth-order Bessel function of the first kind.

We now compare the performance of CURT and a fixed scheduling solution in a period of 300 ms which consists of 10 SFs, numbered by SF 1, 2, $\cdots$, 10. We use the same network scenario with 30 users as described earlier. Each users' UL and DL rate requirements are both set to 10 Mb/s. For the fixed solution, we employ the optimal solution computed off-line for SF 1 in all the 10 SFs. Results under user mobility $v = 1.5$ km/h and $v = 3.0$ km/h are presented in Fig. 2.9 and Fig. 2.10, respectively.[6]. In Fig. 2.9(a) and Fig. 2.10(a), blue and yellow bars represent the numbers of LTE users whose UL and DL rate requirements are

[6]User mobility no more than 3.0 km/h (walking speed) is a common assumption for LTE small cells [5]. High-speed users are usually supported by macro cells in licensed spectrum to avoid frequent handover.



(a) Number of users meeting rate requirements.

(b) Computation time of CURT.

Figure 2.10: Performance of CURT under temporal channel variations for 30 users. User mobility $v = 3.0$ km/h.

both satisfied by CURT and the fixed solution, respectively. We can see that CURT always finds solutions in 1 ms and meets all 30 users' rate requirements; While the fixed solution (optimal for SF 1) expires quickly since SF 2 due to channel variations and can no longer satisfy users' rate requirements. These results indicate that even with low user mobility, it is necessary to have an LTE scheduler that is able to re-compute scheduling solution for each SF in real-time (on ∼1 ms time scale).

2.8 Chapter Summary

In this chapter, we investigated a resource scheduling problem for LTE in unlicensed bands with CSAT-based coexistence paradigm for ensuring fairness to Wi-Fi. We formulated the scheduling problem as an optimization problem of selecting channels for LTE UL and DL transmissions, dividing transmission time on each channel for LTE and Wi-Fi, and allocating RBs on all channels based on LTE users' channel conditions and UL/DL rate requirements. The objective is to minimize LTE's adverse impact on Wi-Fi on each channel. We proved

that this scheduling problem is NP-hard. Then we presented CURT – a novel design of an LTE scheduler for CSAT-based coexistence with Wi-Fi that is able to obtain near-optimal scheduling solution on ∼1 ms time scale. CURT exploits problem decomposition techniques and massive number of cores on low-cost off-the-shelf GPUs to achieve parallel real-time computing. To validate the performance of CURT, we implemented it on NVIDIA GPU/CUDA platform and conducted extensive experiments. Our experimental results demonstrated that CURT can deliver near-optimal scheduling solution on ∼1 ms time scale and meet all of our design expectations.

Chapter 3

An Ultrafast GPU-based Proportional Fair Scheduler for 5G NR

3.1 Introduction

As the new cellular communication standard, 5G NR is designed to cover a wide range of use cases, including broadband human-oriented communications, time-sensitive applications with ultra-low latency, and massive connectivity for Internet of Things [34]. With its wide range of operating frequencies from sub-GHz to 100 GHz [37], the channel coherence time for NR varies greatly. Compared to LTE, which typically operates on bands lower than 3 GHz [41] and with a coherence time over 1 ms, NR is likely to operate on higher frequency range (e.g., 3 to 6 GHz), with much shorter coherence time (e.g., $\sim$200s μs).[1] Further, from application's perspective, 5G NR is expected to support applications with ultra-low latency (e.g., augmented/virtual reality, autonomous vehicles) [42], which may require millisecond or even sub-millisecond scale delay or response time.

To support such diverse service cases and channel conditions, the air interface of NR is designed to be much more flexible and scalable than that of LTE.[2] Specifically, a number

[1] A widely-used definition of coherence time is given by $T_c = 9/16\pi f_m$, where $f_m = v/\lambda$ is the maximum Doppler shift, v is the user speed and λ is the carrier wave length [43]. For instance, with $v = 120$ km/h, on 6 GHz spectrum we have $T_c \approx 260$ μs.

[2] We use the terms 5G NR and NR interchangeably throughout this chapter.

Table 3.1: OFDM numerologies in NR [2].

Numerology	SC spacing	Time slot duration	Time slots per subframe	Suitable bands
0	15 kHz	1000 μs	1	$\leq$ 6 GHz
1	30 kHz	500 μs	2	$\leq$ 6 GHz
2	60 kHz	250 μs	4	$\leq$ 6 GHz
3	120 kHz	125 μs	8	$\leq$ 6 GHz or $\geq$ 24 GHz
4	240 kHz	62.5 μs	16	$\geq$ 24 GHz

of OFDM numerologies are defined for NR [2], allowing a wide range of frequency and time granularities for data transmission (see Table 3.1). Instead of a single transmission time interval (TTI) setting of 1 ms as for LTE, NR allows 4 numerologies (0, 1, 2, 3) for data transmission [39],[3] with TTI varying from 1 ms to 125 μs [35]. Most notably, numerology 3 allows NR to cope with extremely short coherence time and meet the stringent requirement in ultra-low latency applications, where the scheduling resolution should be $\sim$100 μs.

But the new $\sim$100 μs time requirement also poses a new technical challenge to the design of an NR scheduler. To concretize our discussion, we use the most popular *proportional-fair* (PF) scheduling as an example [50, 51, 52, 53, 54]. Within each scheduling time interval (i.e., a TTI), a PF scheduler needs to decide how to allocate frequency-time *resource blocks* (RBs) to users and determine *modulation and coding scheme* (MCS) for each user. The objective of a PF scheduler is to maximize the sum of logarithmic (long-term) average data rates of all users. An important constraint is that each user can only use one MCS (from a set of allowed MCSs) across all RBs that are allocated to her. This problem is found to be NP-hard [52, 53, 54] and has been widely studied in the literature.

Although some of the existing PF schedulers could offer a scheduling solution on a much larger time scale, none of them can offer a solution close to 100 μs. In [51], Kwan *et al.*

[3]Numerology 4 is used for control signaling.

formulated the PF scheduling problem as an integer linear programing (ILP) and proposed to solve it using branch-and-bound technique, which has exponential computational complexity. Some polynomial-time PF schedulers that used efficient heuristics can be found in [52, 53, 54]. We will examine computational complexity and, more importantly, real-time computational time of these schedulers in Section 3.5. A common feature of these PF schedulers (designed for LTE) is that they are all of sequential designs and need to go through a large number of iterations to determine a solution. Although they may meet the scheduling time requirement for LTE (1 ms), none of them comes close to meet the new 100-μs time requirement for 5G NR.

In this chapter, we present a novel design of a PF scheduler using off-the-shelf GPU to achieve a 100-μs scheduling resolution. We call our design "GPF+", which is the abbreviation of GPU-based PF scheduler, and the "+" sign denotes an improved version over our original design (GPF) in [55]. Key ideas of GPF+ are: (i) to decompose the original scheduling problem into a large number of small and independent sub-problems with the same mathematical structure, where each sub-problem can be solved by a small number of iterations; (ii) to identify a promising search sub-space through a cross-entropy (CE)-based intensification approach and select a subset of sub-problems to fit into a GPU platform through random sampling; (iii) to solve selected sub-problems in parallel using the massive processing cores of a GPU and minimize the execution time based on the underlying GPU platform. We summarize our main contributions in this chapter as follows:

- This chapter presents the first design of a PF scheduler that can offer a timing performance under 100 μs, which is required by 5G NR. This design can support NR numerology 0 to 3, which are to be used for data transmission. To our knowledge, GPF+ is also the fastest scheduler for 5G that is based on GPU computing. Our design only uses a commercial off-the-shelf GPU platform and does not require any

expensive customized hardware.

- Our GPU-based design is based on a decomposition of the original optimization problem into a large number of sub-problems through enumerating MCS assignments for each user. We show that for each sub-problem (with a given MCS assignment), the optimal RB allocation problem can be solved exactly and efficiently.

- To reduce the number of sub-problems and fit them into a commercial off-the-shelf GPU, we identify the most promising search sub-space among all sub-problems through a novel CE-based intensification approach. Then a subset of sub-problems are selected from the promising sub-space by random sampling. We show that such an approach can lead to an optimal or near-optimal scheduling solution.

- We implement GPF+ on an off-the-shelf NVIDIA Tesla V100 GPU using the CUDA programming model. Through a number of engineering efforts such as (i) optimizing the operations performed on the GPU processing cores, (ii) minimizing the memory access time based on differences in memory types and locations, and (iii) reducing iterative computations by exploiting techniques such as parallel reduction, we have successfully achieved overall scheduling time under 100 μs for NR macro-cells.

- We conduct extensive experiments to investigate the performance of GPF+ and compare it with three state-of-the-art LTE PF schedulers. Experimental results show that GPF+ can achieve near-optimal PF performance under 100 μs while other schedulers require significantly higher time (ranging from many times to several orders of magnitude) and none of them comes close to meet the 100-μs time requirement.

- By breaking down the scheduling time of GPF+ between the data movement across the host/GPU and the computation on GPU, we show that nearly 50% of the time overhead is incurred by the data movement. This suggests that our GPF+ scheduler

can achieve even better timing performance (e.g., < 50 μs) if a customized GPU system (e.g., integrated host/GPU architectures [79, 80]) is employed.

The rest of the chapter is organized as follows. In Section 3.2, we review related work on using GPU computing for networking and communications. In Section 3.3, we offer a primer of 5G NR air interface. In Section 3.4, we formulate the PF scheduling problem in the context of 5G NR. In Section 3.5, we highlight the challenge in designing a scheduler to meet the 100-μs requirement. In Section 3.6, we present our design ideas of GPF+. The detailed implementation of GPF+ on an NVIDIA Tesla V100 GPU is given in Section 3.7. In Section 3.8, we conduct an experimental study to validate the performance of GPF+. Section 3.9 concludes this chapter.

3.2 Related Work

In the literature, there have been a number of works that studied the application of GPUs in various areas of communications and networking, including MIMO detection [56, 57], beamforming [58, 59], network-layer packet processing [60, 61, 62], channel coding/decoding [63, 64], and MAC-layer scheduling [65, 66].

In [56], the authors proposed FlexCore, which is a MIMO detector design that utilizes multi-processors to evaluate the nodes in a detection search tree in parallel. The efficacy of the proposed detector was validated through implementations on GPU and FPGA. In [57], the authors proposed a soft-output MIMO detector that employs a norm-based ordering strategy for parallel tree search. This design was targeted at a GPU-based implementation. However, in both [56, 57], the proposed search tree-based parallel designs do not fit the problem structure of PF resource scheduling.

The work in [58] addressed real-time computation of digital beamforming weights for massive multi-user (MU)-MIMO using GPU. The design reduces the complexity of singular-value-decomposition (SVD) and decomposes the MU-MIMO beamforming across the channel bandwidth into a large numer of single-user (SU)-MIMO beamforming sub-problems, which are then solved in parallel by GPU cores. In [59], a decentralized beamforming architecture for massive MU-MIMO was proposed which divides the base station (BS) antennas into independent groups and then uses a cluster of GPUs to perform beamforming for each group of antennas using only local channel state information. Unfortunately, neither the problem decomposition technique in [58] nor the grouping method in [59] can be applied to our PF scheduling problem.

In [60], the authors proposed PacketShader, which is a GPU-based software router that utilizes parallelism in packet processing to boost network throughput. In [61], the authors applied GPU to network traffic indexing and were able to achieve an indexing throughput over one million records per second. In [62], the authors designed a packet classifier that is optimized for GPU's memory hierarchy and massive number of cores. All these works focused on network-layer packet processing, which is fundamentally different from the resource scheduling problem we are studying in this chapter.

In [63], the authors designed GPU-based parallel decoders for LDPC channel code. The work in [64] addressed the implementation of a fully parallelized LTE Turbo decoder on GPU. These works employed GPUs for baseband channel coding/decoding and the proposed approaches cannot be applied to solve optimal scheduling problems.

The works in [65, 66] explored the application of GPUs in MAC-layer scheduling and are most relevant to this chapter. Specifically, in [65], the authors studied channel division and resource allocation across multiple unlicensed bands shared between LTE and Wi-Fi. The original optimization problem is decomposed into a moderate number (10s) of sub-problems

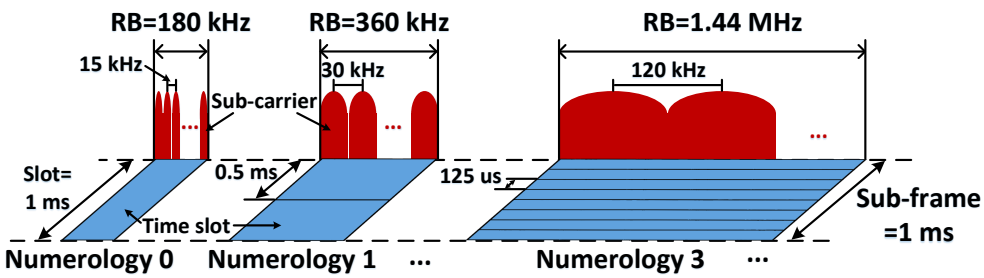


Figure 3.1: A frame structure of NR.

which are solved in parallel on GPU. In contrast, the decomposition technique in this chapter produces a much larger number (e.g., 29^{100}) of sub-problems. Further, our proposed intensification and sampling methods fundamentally differ from the solution approach in [65]. In [66], the authors studied an age-of-information (AoI) minimization problem through resource scheduling in a 5G network. Following a problem relaxation and reformulation, the original AoI minimization problem was transformed into an integer quadratic programming (IQP). Then a metric-based heuristic tailored for GPU implementation was proposed to solve the IQP. Such a metric-based algorithm follows the conventional heuristic design and is completely different from our approach that exploits decomposition and sub-problem selection techniques.

3.3 A Primer on NR Air Interface

To meet the extremely diverse operating conditions, NR has a much more flexible and scalable air interface compared to 4G LTE [67]. A frame structure of NR is illustrated in Figure 3.1. In the frequency domain, NR still employs OFDM and the bandwidth of a channel is divided into a number of sub-carriers (SCs). In the time domain, each frame has 10 ms duration and consists of 10 sub-frames (SFs), each with 1 ms duration. An SF may contain one or multiple time slots. The number of time slots per SF is determined by the OFDM numerology [2].

Table 3.1 shows the SC spacing, the number of time slots per SF, the duration of each time slot, and the suitable frequency bands under each OFDM numerology. In NR, the number of OFDM symbols per time slot is fixed to 14 [2]. Thus the duration of a time slot becomes shorter when the SC spacing is larger. Since numerology 4 will only be used for control signaling [39], we will focus our discussion on numerology 0 to 3 (for data transmission) in this chapter.

A transmission time interval (TTI) is the scheduling time unit (or the scheduling resolution) of the BS scheduler. Based on the service requirements, the duration of a TTI can be configured to one time slot or multiple time slots [34]. For example, under numerology 0, a TTI with a single time slot has 1 ms duration, which is backward compatible to 4G LTE where a TTI is fixed to 1 ms. The same TTI duration can also be achieved by combining two time slots in a TTI under numerology 1 (0.5 ms per time slot). Under numerology 3, a one-slot TTI has 125 μs duration, which enables a very short transmission latency in the user plane. Such a TTI configuration can be employed to meet the timing requirements of delay-sensitive services (e.g., ultra-reliable low-latency communications (URLLC)).

In the frequency domain, the scheduling resolution is one RB, which consists of 12 consecutive SCs. Within each TTI, the BS scheduler needs to decide how to allocate the RBs on the channel to the active users in the cell for the next TTI. Therefore, the channel coherence time should span at least two TTIs.

In each TTI, although an RB is allocated to a single user, a user may be allocated to multiple RBs. Then a key question is what modulation and coding scheme (MCS) should be used for each user. In NR, there are 29 usable MCS levels [36], each representing a combination of modulation and coding techniques.[4] The BS scheduler needs to choose a

[4]More precisely, 31 MCS levels are defined in NR standard, with two of them being reserved, leaving 29 MCS levels available [36].

MCS level for each user which will be used for all its allocated RBs [36].[5] Such a requirement is the same as 4G LTE [68]. One reason is that using different MCS levels across the channel bandwidth cannot offer a significant performance gain, but will incur excessive control signaling overhead [47]. For each user, the optimal choice of MCS level depends on the channel conditions on its allocated RBs. The scheduling decision for each TTI entails the joint RB allocations and MCS selections for all the scheduled users.

3.4 A Formulation of the PF Scheduling Problem

In this section, we present a formulation of the classical PF scheduling problem under 5G NR. Table 3.2 gives notation that we use in this chapter.

3.4.1 Modeling and Formulation

Consider a 5G NR BS and a set $\mathcal{U}$ of users under its service. Denote $U = |\mathcal{U}|$ as the total number of users. For scheduling at the BS, we focus on the downlink (DL) direction (data transmissions from BS to users) and assume a full-buffer model, i.e., there is always data backlogged at the BS for each user. Denote W as the total bandwidth of the DL channel. Under OFDM, radio resource on this channel is organized as a two-dimensional frequency-time resource grid. In the frequency domain, the channel bandwidth is divided into a set $\mathcal{B}$ of RBs, each with bandwidth $W_0 = W/B$, where $B = |\mathcal{B}|$. Due to frequency-selective channel fading, channel condition for a user varies across different RBs. For the same RB, channel conditions from the BS to different users also vary, due to the differences in their geographical locations. In the time domain, we have consecutive TTIs, each with a duration

[5] In each TTI, the transport block (TB) of a user may contain one or two codewords. We follow the convention that each user employs a common MCS for the entire TB [51, 52, 54].

T_0. Scheduling at the BS must be completed within the current TTI (before the start of the next TTI).

Denote $x_u^b(t) \in \{0,1\}$ as a binary variable indicating whether or not user $u \in \mathcal{U}$ is allocated with RB $b \in \mathcal{B}$ in TTI t, i.e.,

$$x_u^b(t) = \begin{cases} 1, & \text{if user } u \text{ is allocated with RB } b \text{ in TTI } t, \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

Since each RB can be allocated to at most one user, we have:

$$\sum_{u \in \mathcal{U}} x_u^b(t) \leq 1, \quad (b \in \mathcal{B}). \tag{3.2}$$

At the BS, there is a set $\mathcal{M}$ of MCS levels that can be used by the transmitter for each user $u \in \mathcal{U}$ in TTI t. When multiple RBs are allocated to a user, only one MCS level, denoted by m ($m \in \mathcal{M}$), can be used for all these RBs. Denote $y_u^m(t) \in \{0,1\}$ as a binary variable indicating whether or not user $u \in \mathcal{U}$ is assigned with MCS $m \in \mathcal{M}$ in TTI t, i.e.,

$$y_u^m(t) = \begin{cases} 1, & \text{if user } u \text{ is assigned with MCS } m \text{ in TTI } t, \\ 0, & \text{otherwise.} \end{cases} \tag{3.3}$$

Since in each TTI t a user u can only use one MCS in $\mathcal{M}$, we have:

$$\sum_{m \in \mathcal{M}} y_u^m(t) \leq 1, \quad (u \in \mathcal{U}). \tag{3.4}$$

On a given RB b, a user u's achievable data rate is determined by the channel condition and the selected MCS. Figure 3.2 shows the spectral efficiencies of the 29 MCS levels defined in NR. In general, a higher MCS level corresponds to a higher spectral efficiency. The highest

Table 3.2: Notation in Chapter 3

Symbol	Definition
$\mathcal{B}$	The set of RBs on the channel
B	$= \lvert\mathcal{B}\rvert$, the total number of RBs
K	The total number of sub-problems solved in each TTI
$\mathcal{M}$	The set of usable MCSs
M	$= \lvert\mathcal{M}\rvert$, the total number of usable MCSs
$\mathcal{U}$	The set of users in the cell
U	$= \lvert\mathcal{U}\rvert$, the total number of users
N_c	The window size (in number of TTIs) for PF scheduling
$q_u^b(t)$	The highest MCS that is feasible to user u on RB b in the t-th TTI
$q_u^{\max}$	The highest MCS that is feasible to user u across all RBs
r^m	The achievable data rate on each RB under MCS m
$r_u^{b,m}(t)$	User u's instantaneous data rate on RB b under MCS m in the t-th TTI
$R_u(t)$	User u's sum achievable data rate in the t-th TTI
$\tilde{R}_u$	User u's long-term average data rate
$\tilde{R}_u(t)$	User u's exponentially smoothed average data rate up to the t-th TTI
$\boldsymbol{v}$	The vector of probabilities for intensification
$x_u^b(t)$	A binary variable to indicate whether or not user u is allocated with RB b in TTI t
$y_u^m(t)$	A binary variable to indicate whether or not user u is assigned with MCS m in TTI t
$\boldsymbol{y}$	The vector that contains all y-variables
γ^*	The optimal objective of OPT-PF
γ	The objective achieved by a $\boldsymbol{y}$ vector
$\boldsymbol{Y}$	The feasible region of $\boldsymbol{y}$
$\mathrm{G}(\boldsymbol{y})$	The function mapping a vector $\boldsymbol{y}$ into the optimal objective value of OPT(Y)
$P(\boldsymbol{y};\boldsymbol{v})$	The PMF used for generating $\boldsymbol{y}$ based on $\boldsymbol{v}$
$\boldsymbol{v}^\gamma$	The (unknown) optimal intensification vector for a specific $\gamma < \gamma^*$
$\widehat{\boldsymbol{v}^\gamma}$	The estimate for $\boldsymbol{v}^\gamma$ returned by Algorithm 2
d^+	The number of nonzero elements in $\widehat{\boldsymbol{v}^\gamma}$

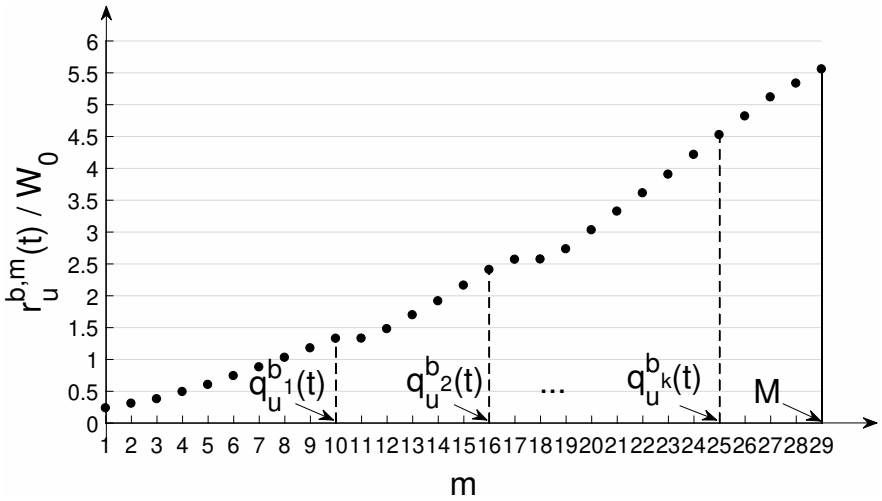


Figure 3.2: Spectral efficiencies of different MCSs ([36], Table 5.1.3.1-1).

achievable data rate per RB is $5.5547W_0$ with MCS 29 [36]. However, the channel condition of user u on RB b may not be able to support all the $M = |\mathcal{M}|$ MCS levels. Denote $q_u^b(t)$ as the highest MCS level that is feasible to user u on RB b in TTI t. The value of $q_u^b(t)$ depends on the channel condition and can be determined from the channel quality indicator (CQI) fed back by user u. Since M is the highest MCS level, we have $q_u^b(t) \leq M$. For a given $q_u^b(t)$, all MCS levels in $\{1, 2, \cdots, q_u^b(t)\}$ will be feasible to user u on RB b. On the other hand, if $q_u^b(t) < M$ and the BS chooses an MCS level $m > q_u^b(t)$ for user u, then its data rate on RB b will drop to zero, due to the excessive bit errors [51, 54]. Clearly, there is a trade-off between the chosen MCS level and the number of RBs that can contribute nonzero data rates.

Denote $r_u^{b,m}(t)$ as user u's achievable data rate on RB b with MCS level m. Then we have

$$r_u^{b,m}(t) = \begin{cases} r^m, & \text{If } m \leq q_u^b(t), \\ 0, & \text{If } m > q_u^b(t), \end{cases} \tag{3.5}$$

where r^m is the achievable data rate per RB with MCS level $m \in \mathcal{M}$. Recall that for each user $u \in \mathcal{U}$, the BS must use the same MCS level $m \in \mathcal{M}$ for all its allocated RBs. Thus

there exists a tension between the achievable data rate per RB and the number of RBs that can actually offer this data rate. As an example, suppose there are five RBs, denoted by b_1, b_2, b_3, b_4 and b_5, that have been allocated to a user u. Suppose $q_u^{b_1}(t) = 10$, $q_u^{b_2}(t) = 16$, $q_u^{b_3}(t) = 18$, $q_u^{b_4}(t) = 21$, and $q_u^{b_2}(t) = 25$. Then based on Figure 3.2, if we choose MCS 10 for this user, all the five RBs can contribute a data rate $r^{10} = 1.3281W_0$. On the other hand, if MCS 21 is chosen, only b_4 and b_5 can contribute a bit rate $r^{21} = 3.6094W_0$. Clearly, a lower MCS level allows more RBs to be eligible to contribute, but at a lower bit rate; while a higher MCS level allows fewer RBs to be eligible to contribute, but at a higher bit rate.

Denote $R_u(t)$ as the sum achievable data rate of user u in TTI t. Under a given scheduling decision consisting of RB allocation and MCS assignment, we can determine $R_u(t)$ as:

$$R_u(t) = \sum_{b \in \mathcal{B}} x_u^b(t) \sum_{m \in \mathcal{M}} y_u^m(t) r_u^{b,m}(t). \tag{3.6}$$

3.4.2 PF Objective Function

We now describe the formulation of the PF objective function. Denote $\tilde{R}_u$ as user u's long-term average data rate. The objective function of the PF scheduling problem is written as [49, 52]:

$$\sum_{u \in \mathcal{U}} \log \tilde{R}_u.$$

PF scheduling is able to achieve a trade-off between the total throughput in a cell and the fairness among users. To maximize the PF objective, a common approach is to maximize following metric for each TTI t [49, 50, 52, 53]:

$$\sum_{u \in \mathcal{U}} \frac{R_u(t)}{\tilde{R}_u(t-1)} \tag{3.7}$$

where $R_u(t)$ is the total achievable data rate scheduled for user u in TTI t (calculated by (3.6)) and $\tilde{R}_u(t-1)$ is user u's average data rate (up to TTI $(t-1)$) that is exponentially smoothed through a sliding window with a size of N_c TTIs. $\tilde{R}_u(t-1)$ is updated as:

$$\tilde{R}_u(t-1) = \frac{N_c - 1}{N_c}\tilde{R}_u(t-2) + \frac{1}{N_c}R_u(t-1)\ . \tag{3.8}$$

It has been shown that such real-time (per TTI) scheduling algorithm can approach optimal PF objective value asymptotically when $N_c \to \infty$ [49].

In this chapter, we adopt this widely-used PF objective function for our scheduler. Putting (3.6) into (3.7), we have:

$$\sum_{u\in\mathcal{U}} \frac{R_u(t)}{\tilde{R}_u(t-1)} = \sum_{u\in\mathcal{U}}\sum_{b\in\mathcal{B}}\sum_{m\in\mathcal{M}} \frac{r_u^{b,m}(t)}{\tilde{R}_u(t-1)} x_u^b(t) y_u^m(t). \tag{3.9}$$

In (3.9), $r_u^{b,m}(t)$ and $\tilde{R}_u(t-1)$ are input parameters and $x_u^b(t)$ and $y_u^m(t)$ are decision variables.

3.4.3 Problem Formulation

Based on the above discussions, the PF scheduling optimization problem for TTI t can be formulated as:

OPT-PF

$$\text{maximize} \quad \sum_{u\in\mathcal{U}}\sum_{b\in\mathcal{B}}\sum_{m\in\mathcal{M}} \frac{r_u^{b,m}(t)}{\tilde{R}_u(t-1)} x_u^b(t) y_u^m(t)$$

$$\begin{aligned} \text{subject to} \quad & \text{RB allocation constraints: (3.2)}, \\ & \text{MCS assignment constraints: (3.4)}, \\ & x_u^b(t) \in \{0,1\}, \quad (u \in \mathcal{U}, b \in \mathcal{B}) \\ & y_u^m(t) \in \{0,1\}, \quad (u \in \mathcal{U}, m \in \mathcal{M}) \end{aligned}$$

In OPT-PF, $r_u^{b,m}(t)$ is a constant for a given $u \in \mathcal{U}$, $b \in \mathcal{B}$, $m \in \mathcal{M}$ and $q_u^b(t)$ and can be read out from Figure 3.2. Recall that $q_u^b(t)$ is a constant and is determined by the CQI in user u's feedback report at TTI $(t-1)$, which we assume is available by the design of an NR cellular network. $\tilde{R}_u(t-1)$ is also a constant since it is computed in TTI $(t-1)$ based on $\tilde{R}_u(t-2)$ and $R_u(t-1)$. The only variables here are $x_u^b(t)$ and $y_u^m(t)$ $(u \in \mathcal{U},\ b \in \mathcal{B},\ m \in \mathcal{M})$, which are binary integer variables. Since we have product terms $x_u^b(t) \cdot y_u^m(t)$ (nonlinear) in the objective function, we employ the *Reformulation-Linearization Technique* (RLT) [24, 69] to linearize the problem.

To do this, define

$$z_u^{b,m}(t) = x_u^b(t) \cdot y_u^m(t). \tag{3.10}$$

Since both $x_u^b(t)$ and $y_u^m(t)$ are binary variables, $z_u^{b,m}(t)$ is also a binary variable and must satisfy the following RLT constraints:

$$z_u^{b,m}(t) \leq x_u^b(t), \quad (b \in \mathcal{B}, u \in \mathcal{U}, m \in \mathcal{M}), \tag{3.11}$$

$$z_u^{b,m}(t) \leq y_u^m(t), \quad (b \in \mathcal{B}, u \in \mathcal{U}, m \in \mathcal{M}). \tag{3.12}$$

By replacing $x_u^b(t) \cdot y_u^m(t)$ with $z_u^{b,m}(t)$ and adding RLT constraints, we have the following

reformulation for OPT-PF, which we denote by OPT-R:

OPT-R

$$\text{maximize} \quad \sum_{u\in\mathcal{U}}\sum_{b\in\mathcal{B}}\sum_{m\in\mathcal{M}} \frac{r_u^{b,m}(t)}{\tilde{R}_u(t-1)} z_u^{b,m}(t)$$

subject to RB allocation constraints: (3.2),

MCS assignment constraints: (3.4),

RLT constraints: (3.11), (3.12),

$$x_u^b(t) \in \{0,1\}, \quad (b \in \mathcal{B}, u \in \mathcal{U})$$

$$y_u^m(t) \in \{0,1\}, \quad (m \in \mathcal{M}, u \in \mathcal{U})$$

$$z_u^{b,m}(t) \in \{0,1\}. \quad (b \in \mathcal{B}, u \in \mathcal{U}, m \in \mathcal{M})$$

OPT-R is an ILP since all variables are binary and all constraints are linear. Commercial optimizers such as the IBM CPLEX [29] can be employed to find optimal solution to OPT-R (optimal to OPT-PF as well), which can be used as a performance benchmark (instead of a real-time solution). Note that ILP is NP-hard in general and is consistent to the fact that the PF scheduling problem is NP-hard [52, 53, 54].

3.5 The Real-Time Challenge

Although it is possible to design an algorithm to find a near-optimal solution to OPT-R, it remains an open problem to find a near-optimal solution in real time. By real time, we mean that one needs to find a scheduling solution for TTI t during TTI $(t-1)$. For 5G NR, we are talking about on the order of $\sim$100 μs for a TTI, which is nearly an order of magnitude shorter than the scheduling resolution in 4G LTE. This requirement comes from the fact

that the shortest one-slot TTI duration allowed for data transmission is 125 μs under NR numerology 3 (refer to Section 3.3). To the best of our knowledge, none of the near-optimal PF schedulers designed for 4G LTE can achieve such a 100-μs time resolution.

To design a PF scheduler for 5G NR under 100 μs, it is important to first understand why existing LTE schedulers are unable to meet such time requirement. PF schedulers designed for LTE can be classified into two categories: (i) metric-based schemes (typically implemented in industry-grade schedulers) that only address RB allocation [48, 70], and (ii) polynomial-time approximation algorithms that address both RB allocation and MCS assignment [52, 53, 54].

Metric-based schedulers such as those surveyed in [48, 70] allocate RBs to users in each TTI by comparing per-user metrics (e.g., the ratio between instantaneous rate and past average rate) on each RB. These schedulers do not address the assignment of MCS. In a BS, an independent adaptive modulation and coding (AMC) module is in charge of assigning MCS for each user [47]. Therefore, metric-based schedulers cannot be used to solve OPT-PF. From optimization perspective, such a decoupled approach cannot achieve near-optimal performance and will result in a loss of spectral efficiency.

On the other hand, polynomial-time heuristics designed for LTE PF scheduler typically follow a *sequential* algorithm design and thus involve a large number of iterations. Although the use of additional CPU cores may help (by utilizing instruction-level parallelism, e.g., pipelining), the actual reduction in computational time remains unclear. In this chapter, we select several state-of-the-art LTE PF schedulers for performance comparison, including algorithms *Alg1* and *Alg2* from [52], the *Unified Scheduling* algorithm from [53], and the *Greedy* algorithm from [54]. Specifically, *Alg1* and *Alg2* first determine the RB allocation without considering the constraints of one MCS per user, and then fix the problem of multiple MCSs per user by selecting the best MCS for each user under the given the RB allocation. The

computational complexity of *Alg1* and *Alg2* is $O(UBM)$. The *Unified Scheduling* algorithm selects user with its associated MCS and adjusts RB allocation iteratively, until a maximum number of $\bar{K}$ users are scheduled in a TTI. It has a complexity of $O(\bar{K}UBM)$. The *Greedy* algorithm employs a similar iterative design and can support scheduling over multiple carriers. It does not restrict the number of scheduled users per TTI and thus has a complexity of $O(U^2BM)$ for scheduling on a single carrier. In terms of timing performance, *Alg1* and *Alg2* are the fastest among these sequential schedulers. Consider a practical NR cell with 100 users, 100 RBs, and 29 levels of MCS. The number of iterations that *Alg1* and *Alg2* need to go through is roughly 2.9×10^5. Our implementation of *Alg1* on an NVIDIA DGX server with an Intel Xeon E5-2698 v4 CPU shows that the computational time of *Alg1* is beyond 800 μs (see Section 3.8.3).

3.6 A GPU-based Real-Time Scheduler Design

3.6.1 Basic Idea and Roadmap

The basic idea of our design is to decompose the original problem OPT-PF into a large number of small and mutually independent sub-problems, with the solution to each sub-problem being a feasible solution to the original problem. Then the final scheduling solution can be determined through solving sub-problems *in parallel* and finding the solution that achieves the best objective value. We employ the GPU platform to achieve our design due to its low cost, small footprint, and massive parallel processing capability. To make this idea work, we need to address the following two questions:

- (i) How to decompose the original problem so that the sub-problems are independent from each other and can be solved in parallel in real time?

- (ii) How to fit the extremely large number of sub-problems into an off-the-shelf GPU platform?

The first question is directly tied to the time complexity of our scheduler. To meet the real-time requirement, we must be able to solve each sub-problem within 100 μs. Thus it is important that the sub-problems are small in size and require very few (sequential) computations to determine the solutions. Also, it is desirable that the sub-problems have the same mathematical structure and can be solved in a *single-instruction multiple-data* (SIMD) manner, i.e., the solution to each sub-problem is computed with the same set of instructions. SIMD computing is particularly suitable for a GPU platform. This question will be addressed in Section 3.6.2.

The second question is regarding the space limitation of a practical GPU platform. Due to the high-dimensional search space of OPT-PF, the problem decomposition will result in an enormous number of sub-problems. Although a modern GPU typically has thousands of processing cores, it is still far from enough to solve all sub-problems in parallel. Therefore, it is necessary to develop techniques to identify a small subset of sub-problems that are most promising to offer optimal or near-optimal solutions and only solve them on the GPU platform. This selection of these subset of problems will be addressed in Section 3.6.3 and 3.6.4.

3.6.2 Decomposition

There exist a number of decomposition techniques for optimizations, with each designed for a specific purpose. For example, in branch-and-bound methods, a tree-based decomposition is used to iteratively break a problem/sub-problem into smaller sub-problems so as to reduce the size of the problem [71]. In dynamic programming, decomposition produces

sub-problems that require to be solved recursively [72]. But these decompositions cannot be readily parallelized and implemented on GPU.

Our proposed decomposition aims to produce a large number of independent sub-problems with the same mathematical structure. Further, each sub-problem should be small and simple enough so that its solution can be computed by GPU within 100 μs. In other words, our decomposition is tailored toward GPU architecture (massive number of processing cores, high efficiency for SIMD computations, and low clock frequency per core). Such a decomposition can be done by fixing a subset of decision variables via enumerating all possibilities. Then for each resulting sub-problem, we only need to solve the remaining subset of variables.

To see how this can be done for our optimization problem, consider OPT-PF, the original problem formulation with two sets of variables x_u^b and y_u^m, where $u \in \mathcal{U}$, $b \in \mathcal{B}$, $m \in \mathcal{M}$. For the ease of exposition, we omit the TTI index t. Recall that variables x_u^b's are for RB allocation while y_u^m's are for MCS assignment. So we can decompose OPT-PF along either x-variable or y-variable. If we decompose along x, then the total number of sub-problems would be U^B. Otherwise, if we decompose along y, then the total number of sub-problems would be M^U. We choose the latter approach, i.e., to decompose along y, which works naturally with our "intensification" technique proposed in Section 3.6.3.

Denote $\boldsymbol{y}$ as a $1 \times (U \cdot M)$ vector containing all binary variables y_u^m, $u \in \mathcal{U}$, $m \in \mathcal{M}$, i.e.,

$$\boldsymbol{y} = [y_1^1, \cdots, y_1^M, y_2^1, \cdots, y_2^M, \cdots, y_U^1, \cdots, y_U^M].$$

Under a given y-variable assignment $\bar{\boldsymbol{y}}$, whose entries $\bar{y}_u^m$, $u \in \mathcal{U}$, $m \in \mathcal{M}$ satisfy constraint

(3.4), OPT-PF degenerates to the following sub-problem:

OPT(Y)

$$\text{maximize} \quad \sum_{u\in\mathcal{U}}\sum_{b\in\mathcal{B}}\sum_{m\in\mathcal{M}} x_u^b \cdot \frac{r_u^{b,m}}{\tilde{R}_u}\bar{y}_u^m$$

subject to RB allocation constraints: (3.1), (3.2)

In the objective function of OPT(Y), only one term in $\sum_{m\in\mathcal{M}} \frac{r_u^{b,m}}{\tilde{R}_u}\bar{y}_u^m$ is larger than zero (due to the MCS constraint (3.4) on $\bar{y}_u^m$). Denote the m for this nonzero $\bar{y}_u^m$ by m_u^*. Then the objective function of OPT(Y) becomes

$$\sum_{u\in\mathcal{U}}\sum_{b\in\mathcal{B}} x_u^b \cdot \frac{r_u^{b,m_u^*}}{\tilde{R}_u}.$$

By interchanging the order of the two summations, OPT(Y) becomes:

$$\text{maximize} \quad \sum_{b\in\mathcal{B}}\sum_{u\in\mathcal{U}} \frac{r_u^{b,m_u^*}}{\tilde{R}_u} x_u^b$$

subject to RB allocation constraints: (3.1), (3.2)

For a given $b \in \mathcal{B}$, there is only one term in the inner summation $\sum_{u\in\mathcal{U}} \frac{r_u^{b,m_u^*}}{\tilde{R}_u} x_u^b$ that can be nonzero, due to the RB allocation constraint (3.2). So $\sum_{u\in\mathcal{U}} \frac{r_u^{b,m_u^*}}{\tilde{R}_u} x_u^b$ is maximized when the x_u^b corresponding to the largest $\frac{r_u^{b,m_u^*}}{\tilde{R}_u}$ across all users is set to 1 while others are set to 0. Intuitively, this means that the optimal RB allocation (under a given y-variable assignment) is achieved when each RB is allocated to a user that has the largest instantaneous rate normalized by its average rate.

We have just shown how to solve each sub-problem involving x-variable (RB allocation) under a given y-variable (MCS) assignment. The computational complexity of solving each

sub-problem is $O(BU)$, if it is solved sequentially. But the sub-problem structure renders itself naturally for parallel computation, i.e., to perform optimal RB allocation for all RBs in parallel. By doing so, the time complexity of solving each sub-problem is reduced to $O(U)$.

3.6.3 Search Intensification

After problem decomposition by enumerating all possible y-variable (MCS) assignments, we have a total of M^U sub-problems (e.g., $M^U = 29^{100}$ when there are 29 MCS levels and 100 users per cell). Obviously, it is impossible to solve 29^{100} (or M^U in general) sub-problems in parallel due to space limitation (number of cores) of a COTS GPU. So the goal is to identify a subset of these problems that are most promising (in terms of achieving an optimal or near-optimal objective value) and only solve these problems in GPU.

We propose an approach to reduce the search space based on the concept of *intensification* in optimizations (see, e.g., [73]). Instead of searching the entire problem space for an optimal solution, we will only focus our search in a much smaller, but most promising sub-space, i.e., a subset of y-assignments where it is most promising to find optimal or near-optimal solutions. After this intensification step, we will further reduce the search space by only considering a fixed number of sub-problems (from the promising sub-space) via random sampling (Section 3.6.4) so as to fit the available GPU cores.

Cross-Entropy-based Intensification Our proposed intensification method is inspired by the so-called *cross-entropy* (CE) method [74, 75]. CE was originally designed to estimate probability of rare events in complex stochastic networks. The same technique can be used to search for optimal or near-optimal solutions to a combinatorial optimization problem, whose optimal/near-optimal solutions can be considered as rare events in the search space. Conventional CE methods generate a single solution sample that has a good probability to

achieve near-optimal objective value. To improve the quality of the final solution (in terms of its closeness to the optimal objective value), we propose to evaluate a large number of solution samples simultaneously by exploiting the massive number of parallel processors in a GPU. This will lead to a much higher probability of finding optimal or near-optimal solutions compared to a single sample in the original CE method.

'For ease of exposition, we reformulate OPT-PF as follows. Denote $\mathrm{G}(\boldsymbol{y})$ as the function that maps a given $\boldsymbol{y}$ into the corresponding optimal objective value of OPT(Y) (see Section 3.6.2). Then OPT-PF is simply:

$$\begin{aligned} &\text{maximize} && \mathrm{G}(\boldsymbol{y}) \\ &\text{subject to} && \boldsymbol{y} \in \boldsymbol{Y} \end{aligned}$$

where $\boldsymbol{Y}$ represents the entire feasible region of $\boldsymbol{y}$ under constraints (3.4).

First of all, we can readily cut off certain part of the search space without losing the global optimum. As discussed in Section 3.4.1, each user has M MCS levels to choose from, with a higher MCS level offering a greater achievable data rate but requiring better channel quality. Recall that for each RB $b \in \mathcal{B}$, q_u^b is the highest MCS level that is feasible to user u. As q_u^b differs for each RB, denote $q_u^{\max} = \max_{b\in\mathcal{B}} q_u^b$ as the highest feasible MCS of user u across all RBs. Then from the problem search space, we can remove all y-assignments in which the selected MCS for some user $u \in \mathcal{U}$ is higher than $q_u^{\max}$ without losing the global optimum. This is because for a specific user u, using an MCS higher than $q_u^{\max}$ will result in a zero data rate on all its allocated RBs. Through this reduction, only the MCSs in $\{1, \cdots, q_u^{\max}\}$ will be considered for each user.

However, the search space is still too large after this reduction. As a further step to reduce the search space, in [55], we chose to consider only the 3 highest feasible MCSs (i.e.,

$\{q_u^{\max}-2, q_u^{\max}-1, q_u^{\max}\}$) for each user following an empirical study. In this chapter, we improve the heuristic in [55] by employing the CE method to find the optimal subset of MCS levels for each user. More formally, denote $\boldsymbol{v}$ as a $1 \times M$ vector, i.e.,

$$\boldsymbol{v} = [v_1, v_2, \cdots, v_M], \tag{3.13}$$

where $v_i \geq 0$ for $i = 1, \cdots, M$ and $\sum_{i=1}^{M} v_i = 1$. While this vector $\boldsymbol{v}$ is common to all users, each user has its own (different) interpretation when using it. For user u, denote p_1^u, p_2^u, $\cdots$, $p_{q_u^{\max}}^u$ as the probabilities of choosing an MCS equal to $q_u^{\max}$, $q_u^{\max}-1$, $\cdots$, 1, respectively. Then we define

$$\begin{cases} p_1^u = v_1 \ , \\ p_2^u = v_2 \ , \\ \quad \cdots \\ p_{q_u^{\max}}^u = \sum\limits_{i=q_u^{\max}}^{M} v_i \ . \end{cases} \tag{3.14}$$

When generating a sample $\boldsymbol{y}$, for user $u \in \mathcal{U}$, we will follow the probabilities p_1^u, p_2^u, $\cdots$, $p_{q_u^{\max}}^u$ defined above to choose the corresponding MCS.[6]

Denote $P(\boldsymbol{y}; \boldsymbol{v})$ as the probability mass function (PMF) for generating $\boldsymbol{y}$ based on the probabilities in $\boldsymbol{v}$. For a given $\boldsymbol{y}$, denote m_u as the MCS level selected for user u, i.e., $y_u^{m_u} = 1$ in $\boldsymbol{y}$. Since the MCS selction for a user is independent from other users, we have:

$$P(\boldsymbol{y}; \boldsymbol{v}) = \prod_{u \in \mathcal{U}} p_{q_u^{\max}-m_u+1}^u \ . \tag{3.15}$$

Denote γ^* as the optimal objective value of OPT-PF. Assume that for a given $\gamma < \gamma^*$ (e.g., $\gamma = 0.99\gamma^*$), there exists an optimal PMF $P(\boldsymbol{y}; \boldsymbol{v}^\gamma)$ with probabilities $\boldsymbol{v}^\gamma$ so that $\mathrm{G}(\boldsymbol{y}) \geq \gamma$

[6] Incidentally, the MCS selection heuristic employed in [55] corresponds to a very special case here when we fix $v_1 = v_2 = v_3 = 1/3$, and $v_4 = \cdots = v_M = 0$ in $\boldsymbol{v}$.

can be guaranteed almost surely. Now the question is: how can we determine this optimal $\boldsymbol{v}^{\gamma}$?

Inspired by the CE method, our approach is to initialize $\boldsymbol{v}$ with a uniform distribution $\boldsymbol{\pi} = [1/M, 1/M, \cdots, 1/M]$ and then iteratively improve the probabilities in $\boldsymbol{v}$ to approach the optimal $\boldsymbol{v}^{\gamma}$. We now explain how this approach works. Following [74, 75], for a given $\gamma < \gamma^*$, the optimal vector $\boldsymbol{v}^{\gamma}$ can be obtained by solving

$$\begin{aligned} \boldsymbol{v}^{\gamma} &= \arg\max_{\boldsymbol{v}} \mathrm{E}_{\boldsymbol{\pi}} \left[I_{\{\mathrm{G}(\boldsymbol{y}) \geq \gamma\}} \cdot \ln P(\boldsymbol{y}; \boldsymbol{v}) \right], \\ &= \arg\max_{\boldsymbol{v}} \sum_{\boldsymbol{y} \in \boldsymbol{Y}} I_{\{\mathrm{G}(\boldsymbol{y}) \geq \gamma\}} \cdot \ln P(\boldsymbol{y}; \boldsymbol{v}) \cdot P(\boldsymbol{y}; \boldsymbol{\pi}), \end{aligned} \tag{3.16}$$

where the expectation $\mathrm{E}_{\boldsymbol{\pi}}$ is taken with respect to $P(\boldsymbol{y}; \boldsymbol{\pi})$; $I_{\{\mathrm{G}(\boldsymbol{y}) \geq \gamma\}}$ is an indicator function and is defined as

$$I_{\{\mathrm{G}(\boldsymbol{y}) \geq \gamma\}} = \begin{cases} 1, & \text{if } \mathrm{G}(\boldsymbol{y}) \geq \gamma, \\ 0, & \text{otherwise}, \end{cases} \tag{3.17}$$

for $\gamma \in \mathbb{R}$.

Unfortunately, it is not possible to determine $\boldsymbol{v}^{\gamma}$ analytically through (3.16). An intuitive way to estimate $\boldsymbol{v}^{\gamma}$ is to generate a set of random samples $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_N$ from $P(\boldsymbol{y}; \boldsymbol{\pi})$ and then solve the problem

$$\arg\max_{\boldsymbol{v}} \frac{1}{N} \sum_{i=1}^{N} I_{\{\mathrm{G}(\boldsymbol{y}_i) \geq \gamma\}} \cdot \ln P(\boldsymbol{y}_i; \boldsymbol{v}). \tag{3.18}$$

This approach will not work either because when γ is very close to γ^*, there will be very few (or zero) samples in $\{\boldsymbol{y}_1, \cdots, \boldsymbol{y}_N\}$ that can satisfy $I_{\{\mathrm{G}(\boldsymbol{y}_i) \geq \gamma\}} = 1$.

Instead of solving (3.18) with a γ very close to γ^* at the beginning, we employ an approach that starts with a γ that is much lower than γ^*. By iteratively increasing γ, we can generate

Algorithm 2 Estimating $\boldsymbol{v}^{\gamma}$

Initialize $\boldsymbol{v}_0 := \boldsymbol{\pi}$; $j = 1$.
for $j \leq J$ **do**
  Initialize set $\Phi := \emptyset$;
  Randomly generate L problem instances of OPT-PF;
  for each problem instance $k = 1, 2, \cdots, L$ **do**
    Compute the optimal objective value $\gamma^*(k)$ of OPT-PF;
    Generate N random samples $\boldsymbol{y}_1(k), \cdots, \boldsymbol{y}_N(k)$ from $P(\boldsymbol{y}; \boldsymbol{v}_{j-1})$;
    Compute the achieved objective values $\gamma_i(k) := \mathrm{G}(\boldsymbol{y}_i(k))$ and ratios $\beta_i(k) := \gamma_i(k)/\gamma^*(k)$ for $i = 1, \cdots, N$;
    Insert the pairs $(\boldsymbol{y}_i(k), \beta_i(k))$, $i = 1, \cdots, N$ into set Φ;
  end for
  Sort the $L \cdot N$ pairs in Φ in non-decreasing order of $\beta_i(k)$;
  Find the $(1-\rho)$-quantile $\bar{\beta}^j$ of the sorted ratios $\beta_i(k)$'s;
  Compute $\bar{\boldsymbol{v}}_j$ through (3.21);
  Update $\boldsymbol{v}_j := \alpha \bar{\boldsymbol{v}}_j + (1-\alpha)\boldsymbol{v}_{j-1}$;
  $j := j + 1$;
end for
return $\widehat{\boldsymbol{v}^{\gamma}} := \boldsymbol{v}_J$;

new samples and improve the estimation of $\boldsymbol{v}^{\gamma}$. Our algorithm to implement this approach is given in Algorithm 2.

At the beginning of the algorithm, we initialize $\boldsymbol{v}_0$ with the uniform distribution $\boldsymbol{\pi}$. Then, the algorithm executes a total of J iterations to produce a sequence of estimations $\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_J$. In each iteration $j = 1, \cdots, J$, the quality of estimation $\boldsymbol{v}_j$ improves in the sense that the probability of $\mathrm{G}(\boldsymbol{y}) \geq \gamma$ subject to $\boldsymbol{y} \sim P(\boldsymbol{y}; \boldsymbol{v}_j)$ increases. Finally, the output estimation from Algorithm 2 is $\widehat{\boldsymbol{v}^{\gamma}} = \boldsymbol{v}_J$.

Inside each iteration of j, we randomly generate L problem instances of OPT-PF. The constant parameters $r_u^{b,m}$'s and $\tilde{R}_u$'s in each problem instance $k = 1, \cdots, L$ are from an independent TTI, such that the L problem instances are uncorrelated. For each instance k, we first compute the optimal objective value $\gamma^*(k)$ of OPT-PF (using an optimizer such as IBM CPLEX [29] to solve OPT-R). Then, a batch of N sub-problems $\boldsymbol{y}_i(k), i = 1 \cdots, N$ are randomly produced from PMF $P(\boldsymbol{y}; \boldsymbol{v}_{j-1})$ (with the vector $\boldsymbol{v}_{j-1}$ from the last iteration). For

each sample $\boldsymbol{y}_i(k)$, we compute its achieved objective value $\gamma_i(k) = \mathrm{G}(\boldsymbol{y}_i(k))$ and the ratio $\beta_i(k) = \gamma_i(k)/\gamma^*(k)$. Next, the obtained $L \cdot N$ sample pairs $(\boldsymbol{y}_i(k), \beta_i(k))$ are sorted with respect to $\beta_i(k)$ in non-decreasing order.

Denote $\bar{\beta}^j$ as the $(1-\rho)$-quantile of the sorted ratios $\beta_i(k)$'s in iteration j, where ρ is a small positive value within $(0, 1)$ (e.g., 0.1%). So we have $\rho \cdot L \cdot N$ samples with their $\beta_i(k)$'s greater than $\bar{\beta}^j$. Then just as (3.18), we determine the following vector of probabilities

$$\bar{\boldsymbol{v}}_j = \arg\max_{\boldsymbol{v}} \frac{1}{LN} \sum_{k=1}^{L} \sum_{i=1}^{N} I_{\{\beta_i(k) \geq \bar{\beta}^j\}} \cdot \ln P(\boldsymbol{y}_i(k); \boldsymbol{v}), \tag{3.19}$$

where $I_{\{\beta_i(k) \geq \bar{\beta}^j\}}$ is an indicator function similar to (3.17). Based on (3.15), (3.19) is a convex program. So its optimal solution can be found by solving

$$\frac{1}{LN} \sum_{k=1}^{L} \sum_{i=1}^{N} I_{\{\beta_i(k) \geq \bar{\beta}^j\}} \cdot \nabla_{\boldsymbol{v}} \ln P(\boldsymbol{y}_i(k); \boldsymbol{v}) = \mathbf{0}, \tag{3.20}$$

where $\nabla_{\boldsymbol{v}}$ denotes taking derivative with respect to $\boldsymbol{v}$, and $P(\boldsymbol{y}_i(k); \boldsymbol{v})$ is given in (3.15). This gives us

$$\bar{\boldsymbol{v}}_j = \frac{\sum_{k=1}^{L} \sum_{i=1}^{N} I_{\{\beta_i(k) \geq \bar{\beta}^j\}} \boldsymbol{n}_i(k)}{U \sum_{k=1}^{L} \sum_{i=1}^{N} I_{\{\beta_i(k) \geq \bar{\beta}^j\}}}, \tag{3.21}$$

where $\boldsymbol{n}_i(k)$ is an $1 \times M$ vector, i.e., $\left[n_i^1(k), n_i^2(k), \cdots, n_i^M(k)\right]$, with each entry $n_i^m(k)$ representing the number of users that are assigned with their m-th highest feasible MCS level in the sample $\boldsymbol{y}_i(k)$. Note that $\sum_{m=1}^{M} n_i^m(k) = U$.

Finally, we employ an exponential moving averaging to update $\boldsymbol{v}_j$, i.e.,

$$\boldsymbol{v}_j = \alpha \bar{\boldsymbol{v}}_j + (1-\alpha) \boldsymbol{v}_{j-1}, \tag{3.22}$$

where α is a small positive value within $(0, 1)$. After obtaining $\boldsymbol{v}_j$, we proceed to the next

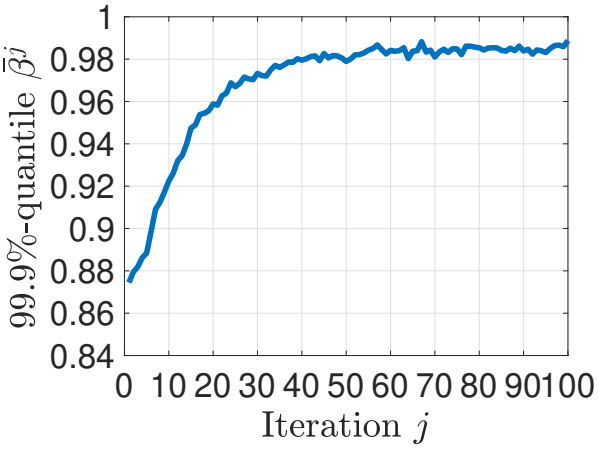


Figure 3.3: $\bar{\beta}^j$ as a function of iterations j in Algorithm 2.

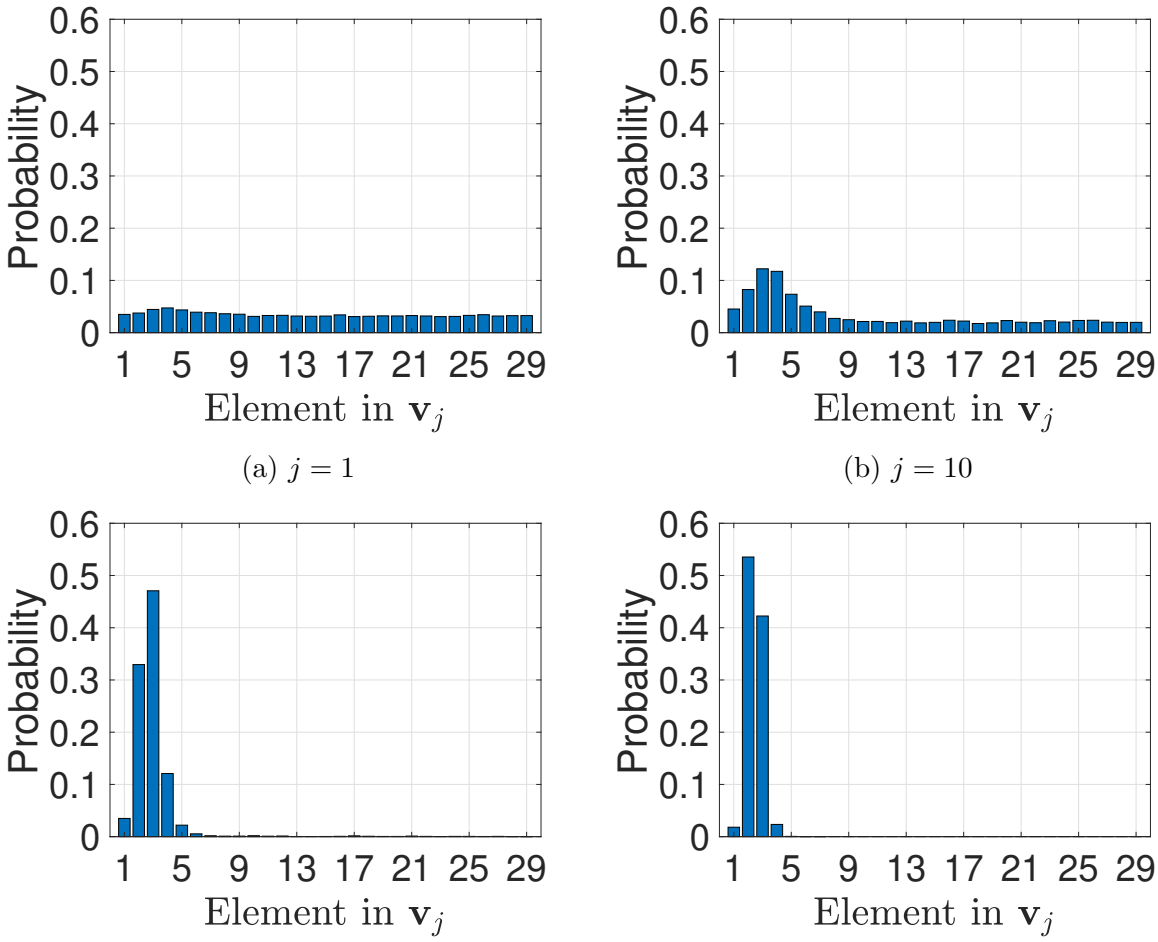
iteration $(j+1)$ or return $\widehat{\boldsymbol{v}^\gamma} = \boldsymbol{v}_J$ when $j = J$.

An Example We now use an example to illustrate the efficacy of our proposed approach. Consider an NR cell with a BS and 100 users. The number of RBs on the channel is $B = 100$. Each user may choose from the $M = 29$ MCS levels shown in Figure 3.2. Other network settings are given in Section 3.8.2.

We run Algorithm 2 for $J = 100$ iterations, with $L = 10$ problem instances per iteration. Under each instance, $N = 10,000$ random samples $\boldsymbol{y}_i(k)$ are generated. A proportion $\rho = 0.1\%$ is used for determining $\bar{\beta}^j$. The soft updating parameter is $\alpha = 0.5$.

Figure 3.3 shows the value of $\bar{\beta}^j$ as a function of the number of iterations, where $\bar{\beta}^j$ is the 99.9%-quantile of the $L \cdot N$ ratios $\beta_i(k)$ in each iteration j. We can see that $\bar{\beta}^j$ steadily improves (increases) as j increases. When $j = J = 100$, $\bar{\beta}^j$ reaches 0.99. This demonstrates that Algorithm 2 can effectively estimate (3.18) when J is sufficiently large.

Figure 3.4 shows the value (probability) of each element in $\boldsymbol{v}_j$ after 1st, 10-th, 50-th, and 100-th iteration, respectively. In Figure 3.4a, we can see that $\boldsymbol{v}_j$ is close to the uniform distribution $\boldsymbol{\pi}$ after one iteration. But as the number of iterations increases in Figures 3.4b



(a) $j = 1$

(b) $j = 10$

(c) $j = 50$

(d) $j = 100$

Figure 3.4: $\boldsymbol{v}_j$ after the j-th iteration in Algorithm 2.

to 3.4d, the probability mass distribution in $\boldsymbol{v}_j$ tends to concentrate over the region of the first few highest MCS levels. □

The results in the above example suggests that we only need to focus our search on those MCS with non-negligible probability values. For example, in Figure 3.4d, only the first 4 elements in $\boldsymbol{v}_j$ have probabilities greater than 0.001. So we can set elements 5 to 29 in v_i to 0 and normalize the first 4 elements (i.e., making the sum of the first 4 elements to 1). We have a new vector $\widehat{\boldsymbol{v}^\gamma} = [0.018, 0.536, 0.423, 0.023, 0, 0, \cdots, 0]$. With this $\widehat{\boldsymbol{v}^\gamma}$, we need to consider no more than 4 highest MCS levels for each user by following this distribution in (3.14). This is a much smaller search space than considering all 29 MCS levels for each user. Note that this CE-based intensification approach is able to identify a promising search sub-space much more accurately than the simple heuristic proposed in [55].

For a given macro-cell, we execute Algorithm 2 (and the zeroing + normalization process) for each population size U (e.g., 1, 2, $\cdots$, 100) to obtain the corresponding intensification vector $\widehat{\boldsymbol{v}^\gamma}$. This is done offline and the result ($\widehat{\boldsymbol{v}^\gamma}$'s under different U) is stored in a lookup table at the BS.

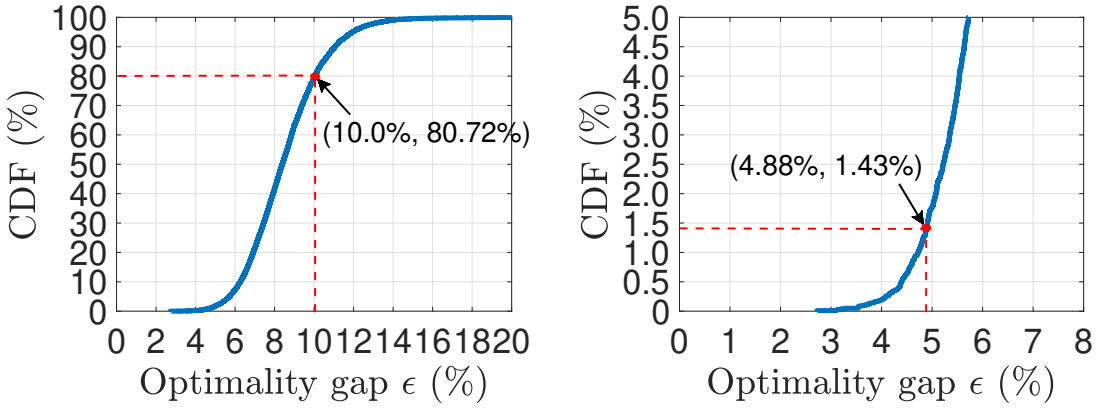
3.6.4 Sampling and Choosing the Best Solution

For scheduling in a TTI, we perform the following steps:

1. **Pre-processing** Generate PMF $P(\boldsymbol{y}|\widehat{\boldsymbol{v}^\gamma})$ based on $\widehat{\boldsymbol{v}^\gamma}$ (according to U in the cell) and each user u's channel condition (i.e., $q_u^{\max}$'s);

2. **Sampling** Randomly generate K samples $\boldsymbol{y}_i$, $i = 1, \cdots, K$ based on $P(\boldsymbol{y}|\widehat{\boldsymbol{v}^\gamma})$;

3. **Finding the best solution**



(a) CDF for $\epsilon \in [0, 20\%]$ (b) CDF for $\epsilon \in [0, 8\%]$

Figure 3.5: CDF as a function of optimality gap (ϵ) based on $10,000$ samples.

- For each $\boldsymbol{y}_i$, $i = 1, \cdots, K$, determine the optimal RB allocation (x-assignment in OPT(Y)) and the achieved objective value;
- Among the K samples $\boldsymbol{y}_i$, $i = 1, \cdots, K$, find the sample $\boldsymbol{y}_i$ that achieves the highest objective value;

4. **Scheduling** Use the best $\boldsymbol{y}_i$ and the corresponding optimal RB allocation (x-assignment) for scheduling in the current TTI.

An important question to ask is in Step 2, how large K should one choose? It turns out that K does not need to be very large. This is good news but rather counter-intuitive. In the following paragraphs, we elaborate how this is possible.

Assume that each sample $\boldsymbol{y}_i$ has a probability $p_{1-\epsilon}$ to achieve a $(1-\epsilon)$-optimal objective value, where ϵ is a very small value. Since all the K samples are generated independently following the same PMF, $p_{1-\epsilon}$ is identical for these sample. Denote $P_{1-\epsilon,K}$ as the probability that at least one out of the K samples can achieve $(1-\epsilon)$-optimal. Then we have:

$$P_{1-\epsilon,K} = 1 - (1 - p_{1-\epsilon})^K. \tag{3.23}$$

(3.23) implies that even for a relatively small $p_{1-\epsilon}$, one can still achieve a large $P_{1-\epsilon,K}$.

In our implementation in Section 3.7 with an NVIDIA Tesla V100 GPU, we generate only $K = 320$ samples per TTI to fit the underlying GPU cores as well as to meet timing requirements. Our goal is to achieve $P_{1-\epsilon,K} \geq 99\%$. Based on (3.23), we only need $p_{1-\epsilon} \geq 1 - \sqrt[320]{1-99\%} = 1.43\%$. We now use empirical results to investigate what level of optimality (i.e., $1-\epsilon$) we are able to achieve with this $p_{1-\epsilon} = 1.43\%$.

Consider a network setting with 25 users in the cell, 100 RBs, and 29 usable MCS levels. We generate a total of 10,000 random $\boldsymbol{y}$ samples based on the intensification vector $\widehat{\boldsymbol{v}^{\gamma}}$ for $U = 25$ as described in Section 3.6.3. Then we compute the optimality gap ϵ (in %) of each sample $\boldsymbol{y}$. The cumulative distribution functions (CDF) of ϵ is given in Figure 3.5. First, Figure 3.5(a) shows that 80.72% of the problems generated based on $\widehat{\boldsymbol{v}^{\gamma}}$ can achieve optimality gap within 10%. This demonstrates that the proposed CE-based intensification approach is effective in identifying a promising search sub-space for $\boldsymbol{y}$. In Figure 3.5(b), we magnifying a portion of Figure 3.5(a) for small values of ϵ. We find that for $p_{1-\epsilon} = 1.43\%$, the corresponding ϵ is 4.88%, meaning that the optimality gap is within 4.88% (or 95.12%-optimal). In other words, by generating only $K = 320$ samples, we have a overall probability $P_{1-\epsilon,K} = 99\%$ to achieve $1-\epsilon = 95.12\%$-optimality.

3.7 Implementation

This section presents our implementation of the design ideas in Section 3.6 on an off-the-shelf GPU platform.

3.7.1 Why GPU

For the purpose of implementing our scheduler design, GPU has a number of advantages over multi-core CPU, FPGA, and ASIC. First and foremost, our design requires a large-scale SIMD processing capability to address hundreds of sub-problems concurrently in each TTI. In this regard, GPU is superior to multi-core CPU and FPGA. It is well known that CPUs are optimized for multiple-instruction multiple-data (MIMD) processing but fall short of large-scale SIMD. On the other hand, although FPGA also offers a high level of parallelism, it is still not comparable to GPU. Second, an important advantage of GPU over FPGA and ASIC is its high efficiency for floating-point operations, which is necessary for computing scheduling solutions. Third, commercial GPUs come with highly flexible programming tools such as NVIDIA CUDA, which allow much faster algorithm development cycles compared to FPGA and ASIC. Also, any change or update to a GPU-based algorithm can be easily done with software. Such a modification is impossible for an ASIC after the hardware is made. Finally, GPUs are low-cost (for NR BSs) and are available off-the-shelf. In contrast, the time and monetary cost for developing an ASIC will be much higher.

3.7.2 Overall Architecture

We implement GPF+ using an NVIDIA Tesla V100 GPU [78] based on the CUDA programming model [25]. A Tesla V100 GPU consists of 80 streaming multi-processors (SMs), with eash SM containing 64 CUDA cores (i.e., 5120 cores in total). The BS host is equipped with an Intel Xeon E5-2698 v4 CPU. It is assumed that at the beginning of each TTI, the input data to the scheduler (including parameters $r_u^{b,m}$ and $\tilde{R}_u$) are stored in the host (CPU) memory. Data communication between the GPU and the host goes through a PCIe 3.0 interface.

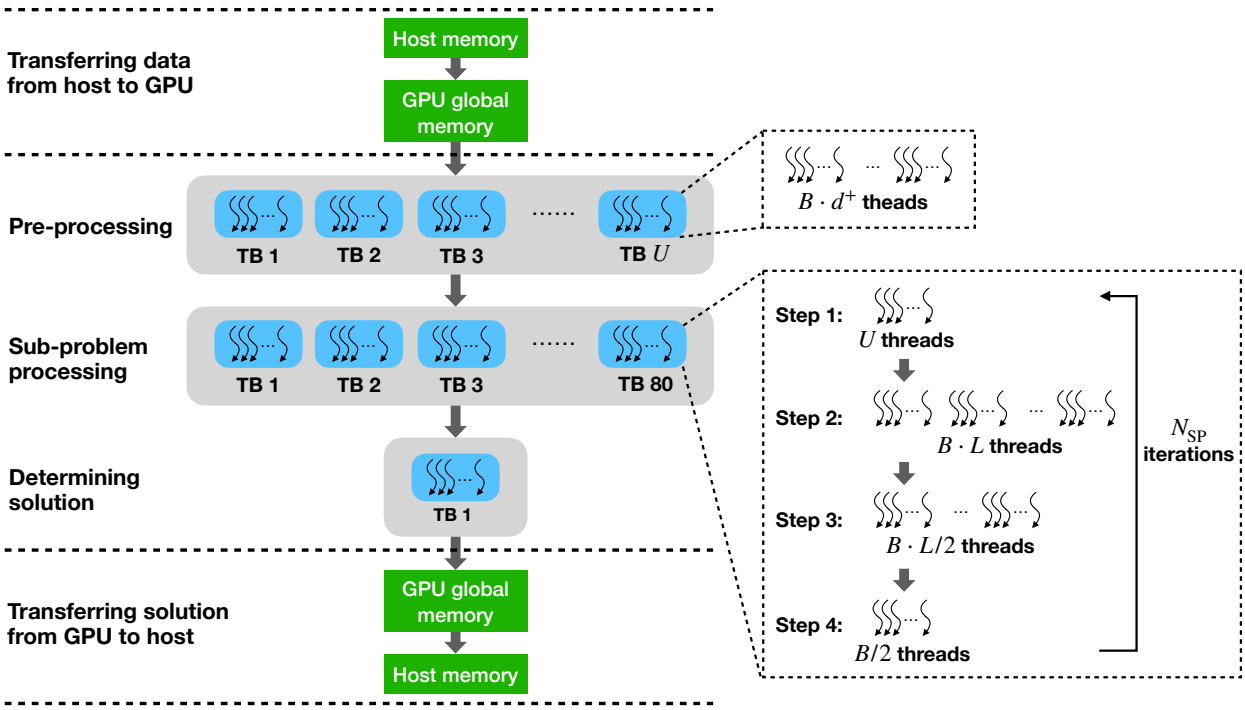


Figure 3.6: A flow chart illustrating our implementation.

Figure 3.6 illustrates the data flow process in our implementation. In each TTI, we need to perform the following tasks across the host/GPU to determine a scheduling solution. The first step is to transfer the required input data from the host to the GPU global memory. Then three CUDA kernels[7] are executed in serial at the GPU to compute the scheduling solution (both RB allocation and MCS selection solution). The first kernel performs pre-processing on the input data. The second kernel generates and solves K sub-problems. Given that a Tesla V100 GPU has 80 SMs, we employ a one-dimensional array of 80 thread blocks (TBs) for this kernel so that each SM can be assigned with a single TB without queuing. Each TB addresses $N_{\text{SP}} = 4$ samples in serial. Thus in each TTI, a total of $K = 80 \times 4 = 320$ sub-problems are processed by GPU. After the K samples are solved, the third kernel compares their achieved objective values and find the best scheduling solution. Finally, this solution is transferred from the GPU back to the host (CPU) memory.

[7]Under CUDA, a kernel represents a function that is executed by an array of threads on GPU.

3.7.3 Some Details

Transfer Input Data to GPU Based on our discussion in Section 3.6.3, only a small subset of MCS levels needs to be considered for each user. Denote d^+ as the number of nonzero entries in $\widehat{\boldsymbol{v}^\gamma}$. For instance, our results in Section 3.8.3 show that d^+=5, 4, 4 and 4 under U=25, 50, 75 and 100, respectively. Thus for each user $u \in \mathcal{U}$, we only need to transfer the parameters $r_u^{b,m}$ corresponding to the MCS levels $q_u^{\max}$, $q_u^{\max}-1$, $\cdots$, $q_u^{\max}-d^++1$ from the host to GPU. This significantly reduces the amount of time for data communication. In addition, we also need to send each user's average data rate $\tilde{R}_u$ to GPU. For instance, with 100 users, we have $d^+=4$. The data to be transferred to GPU includes $r_u^{b,m}$, $m \in \{q_u^{\max}-3, \cdots, q_u^{\max}\}$, $b \in \mathcal{B}$, $u \in \mathcal{U}$ and $\tilde{R}_u$, $u \in \mathcal{U}$. Then in each TTI, the size of the transferred data is 160 KB for $r_u^{b,m}$ plus 0.4 KB for $\tilde{R}_u$ (with float data type).

Pre-processing As described in Section 3.6.2, solving each sub-problem requires computing the ratios $r_u^{b,m}/\tilde{R}_u$. If we perform these computations while searching for the optimal RB allocation under each sub-problem, there will be multiple rounds of memory access to the parameters $r_u^{b,m}$ and $\tilde{R}_u$, along with the division operations. This will result in a high delay in the sub-problem processing. To mitigate issue, we employ a pre-processing kernel to compute these ratios concurrently. As shown in Figure 3.6, this kernel is executed by a one-dimensional array of U TBs, with each TB containing $B \cdot d^+$ threads. Specifically, the threads of each TB computes $r_u^{b,m}/\tilde{R}_u$, $m \in \{q_u^{\max}-d^++1, \cdots, q_u^{\max}\}$, $b \in \mathcal{B}$ for a specific user $u \in \mathcal{U}$ in parallel. This allows memory coalescing on the GPU global memory, i.e., allowing multiple memory accesses (to $r_u^{b,m}$ and $\tilde{R}_u$) and division operations be combined into a single transaction.

Generate and Solve K Sub-problems The second kernel consists of a one-dimensional array of 80 TBs, with a maximum 1,024 threads per TB. In each TTI, this kernel addresses

a total of $K = 320$ sub-problems through $N_{\text{SP}} = 4$ sequential iterations. Each of the 80 TBs will address a single sub-problem per iteration, and all TBs will be executed in parallel (i.e., 80 sub-problems are processed per iteration).

On a particular TB, each iteration involves four steps to generate and solve a sub-problem (see Figure 3.6). Details of these steps are described as follows:

- *Step 1 (Generate a Sub-Problem)* This step generates a sub-problem by randomly generating an MCS level for each user $u \in \mathcal{U}$ following the PMF $P(\boldsymbol{y}|\widehat{\boldsymbol{v}^{\gamma}})$. Denote m_u as the MCS level selected for user u. This MCS selection for each user is done in parallel using U threads.

- *Step 2 (Find Candidate Users for RB Allocation)* This step aims to find a number Z of candidate users from $\mathcal{U}$ for the allocation of each RB $b \in \mathcal{B}$. The purpose is to accelerate the process of finding the optimal RB allocation by concurrently evaluating multiple users for each RB. Specifically, we first uniformly divide the users in $\mathcal{U}$ into Z sub-groups. Then for each RB b, we use Z threads to find the user who has the highest ratio $r_u^{b,m_u}/\tilde{R}_u$ (refer to Section 3.6.2) within each sub-group in parallel. This operation is performed concurrently for all RBs. In this implementation, we set $Z = 10$ when there are $B = 100$ RBs on the channel. Thus this step requires $B \cdot Z = 1,000$ threads. Since the ratios $r_u^{b,m_u}/\tilde{R}_u$ have already been computed by the pre-processing kernel, they can be directly read out from the global memory when needed. The user indexes of the $B \cdot Z$ candidate users and their ratios $r_u^{b,m_u}/\tilde{R}_u$ are stored into the shared memory.

- *Step 3 (Determine Optimal RB Allocation)* Given the $B \cdot Z$ candidate users, this step determines the best user for the allocation of each RB $b \in \mathcal{B}$. This involves comparing the ratios $r_u^{b,m_u}/\tilde{R}_u$ among the Z candidate users per RB. To reduce the number of

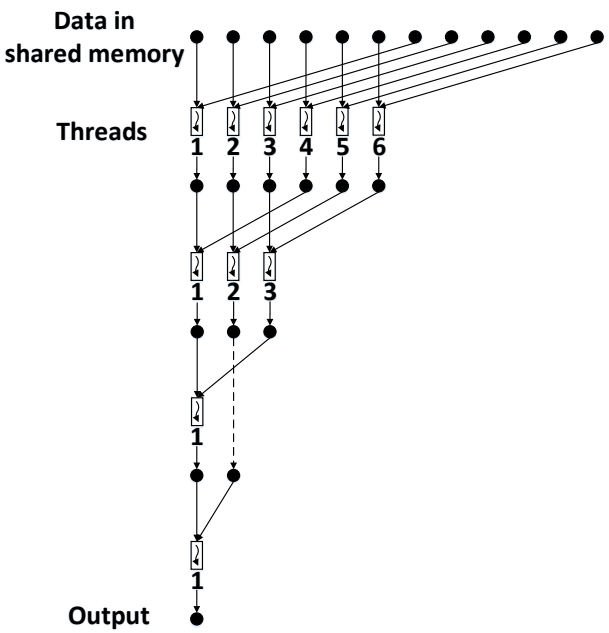


Figure 3.7: An illustration of parallel reduction.

iterations in this step, we employ the *parallel reduction* technique in shared memory [76]. Figure 3.7 illustrates this technique. Specifically, we use $Z/2$ threads to find the best user for each RB. Doing this in parallel across all RBs requires $B \cdot Z/2$ threads.

- *Step 4 (Compute Objective Value).* This step computes the objective value achieved by the optimal RB allocation obtained in Step 3. To do so, we need to find the sum of B elements in the shared memory. Again, this can be accelerated by parallel reduction with $B/2$ threads. As a result, this step only requires $\log_2(B)$ iterations. Then the scheduling solution (both x and y-assignments) along with the achieved objective value are stored into the global memory.

Steps 1-4 will are executed for $N_{\text{SP}} = 4$ iterations by each of the 80 TBs.

Determine Output Scheduling Solution After the second kernel returns, we have $K = 320$ sub-problem solutions and their achieved objective values stored in the global memory. Then we create a new thread block (with 1,024 threads) to find the solution that

achieves the highest objective value. This will be the output solution of the scheduler. This step is also done through parallel reduction.

Transfer Solution from GPU to Host After we determine the output scheduling solution, we transfer it from the GPU global memory back to the BS host memory. This solution will be used for scheduling in the current TTI.

3.8 Experimental Validation

3.8.1 Experiment Platform

Our experiment is done on an NVIDIA DGX server using an Intel Xeon E5-2698 v4 CPU (2.20 GHz) and an NVIDIA Tesla V100 GPU (32 GB memory). Data communication between CPU and GPU goes through a PCIe 3.0 X16 slot with default configuration. Implementation on GPU is based on the NVIDIA CUDA (version 10.2) programming model.

3.8.2 Settings

We consider an NR cell with a BS and a number of users. The user population size U is chosen from $\{25, 50, 75, 100\}$.[8] The number of available RBs is $B = 100$. Assume that a set of $M = 29$ MCSs shown in Figure 3.2 is available to each user. Numerology 3 (refer to Table 3.1) in NR is considered, where the duration of a TTI is 125 μs. The PF parameter N_c is set to 100 TTIs. The full-buffer traffic model is employed.

For wireless channels, we consider the block-fading channel model for both frequency and time [77]. User mobility is captured by independent variations of channel conditions ($q_u^b(t)$'s)

[8] $U = 100$ active users per cell covers all typical deployment scenarios considered in 3GPP (e.g., indoor hotspot, dense urban, rural, etc.) [34].

across TTIs. To model large-scale fading effect, the highest feasible MCS level across all RBs is higher for users that are closer to the BS and is lower for cell-edge users. For frequency-selective fading effect, we consider the worst-case scenario where parameters $q_u^b(t)$'s across all RBs are uncorrelated and randomly generated for each user. Such setting is useful to examine the robustness of GPF+ under extreme operating conditions.

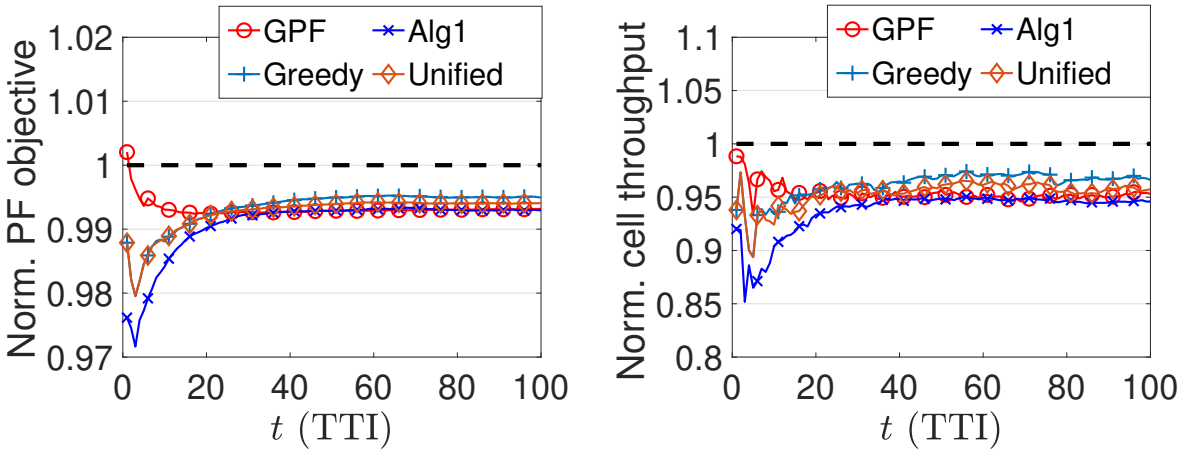
3.8.3 Results

Computing the Intensification Vectors $\widehat{\boldsymbol{v}^\gamma}$ The intensification vectors $\widehat{\boldsymbol{v}^\gamma}$ under $U \in \{25, 50, 75, 100\}$ are computed following Section 3.6.3. Results are presented as follows:

- $U = 25$: $\widehat{\boldsymbol{v}^\gamma} = [0, 0.007, 0.35, 0.505, 0.135, 0.003, 0, \cdots, 0]$;
- $U = 50$: $\widehat{\boldsymbol{v}^\gamma} = [0, 0.254, 0.638, 0.106, 0.002, 0, \cdots, 0]$;
- $U = 75$: $\widehat{\boldsymbol{v}^\gamma} = [0.004, 0.440, 0.527, 0.029, 0, 0, \cdots, 0]$;
- $U = 100$: $\widehat{\boldsymbol{v}^\gamma} = [0.018, 0.536, 0.423, 0.023, 0, 0, \cdots, 0]$.

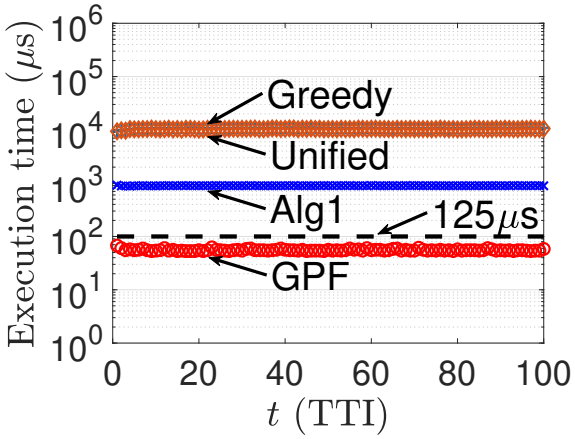
Benchmark and State-of-the-Art Comparisons For performance benchmarking, we employ the IBM CPLEX Optimizer (version 12.7.1) [29] to find the optimal solution to OPT-R. In addition to the optimal solution obtained by CPLEX, we also include state-of-the-art PF schedulers *Alg1* from [52], the *Unified* algorithm from [53], and the *Greedy* algorithm from [54] in our performance comparison. We set the maximum number of scheduled users per TTI to 20 for the *Unified* algorithm. Since these schedulers are essentially sequential algorithms, we implement them on CPU so as to optimize their timing performance.

Results for $U = 25$ Users We first examine the performance of GPF+ under a user population size $U = 25$. We consider two important performance metrics for PF schedulers,



(a) Normalized PF objective

(b) Total cell throughput



(c) Execution time

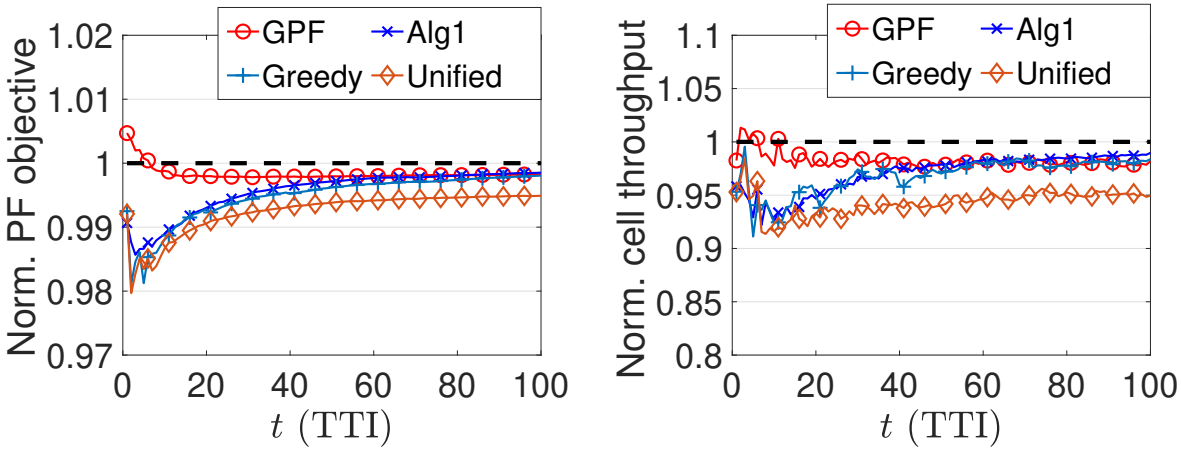
Figure 3.8: GPF+'s performance vs. other state-of-the-art PF schedulers for $U = 25$ users.

including the PF objective $\sum_u \log_2(\tilde{R}_u(t))$, and the sum average throughput in the cell $\sum_u \tilde{R}_u(t)$. Figure 3.8(a) and 3.8(b) show the PF and sum cell throughput performance of GPF+ and the other three PF schedulers for 100 consecutive TTIs, respectively. In these figures, we take the ratio between the performance metric (PF or cell throughput) achieved by a scheduler and that achieved by the optimal solution from CPLEX. Note that there are instances where the ratio is greater than one. This is because CPLEX's solution is optimal with respect to the per-TTI objective (3.7), but not the long term PF criterion that we consider for comparison. Clearly, GPF+ achieves near-optimal performance in terms of both the PF and cell throughput metrics and is comparable to the other three PF schedulers.

In Figure 3.8(c), we compare the execution time of GPF+ and the other three PF schedulers and investigate whether GPF+ is able to meet the 125 μs real-time requirement (the main objective of this chapter). The total execution time of GPF+ includes the time cost for data transferring between the host/GPU and the computation time at GPU (refer to Figure 3.6). We see that GPF+'s execution time is under 100 μs and well below the 125 μs real-time requirement in all TTIs. The mean and standard deviation of GPF+'s execution time are 55.76 μs and 2.35, respectively. On the other hand, *Alg1*, which is the fastest among the existing PF scheduling algorithms, has a mean computation time of 896.74 μs, which is far beyond the 125 μs timing requirement. The computation time of CPLEX for finding optimal solution is not shown because it is too large to fit in the scale of the figure. The average computation time of CPLEX is 2.01 s.

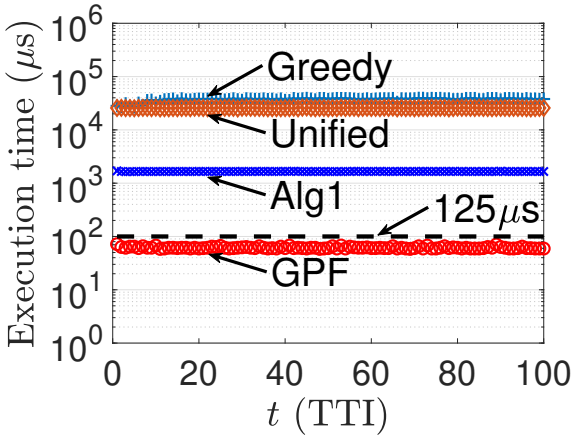
Additional Results for $U = 50$, 75, and 100 Users Now we present additional results for $U = 50$, 75 and 100 in Figures 3.9, 3.10 and 3.11, respectively. In all cases, we see that GPF+ achieves near-optimal PF objective and cell throughput performance and is comparable to the other three PF schedulers.

Figures 3.9(c), 3.10(c) and 3.11(c) show the execution time of GPF+ and compare it to



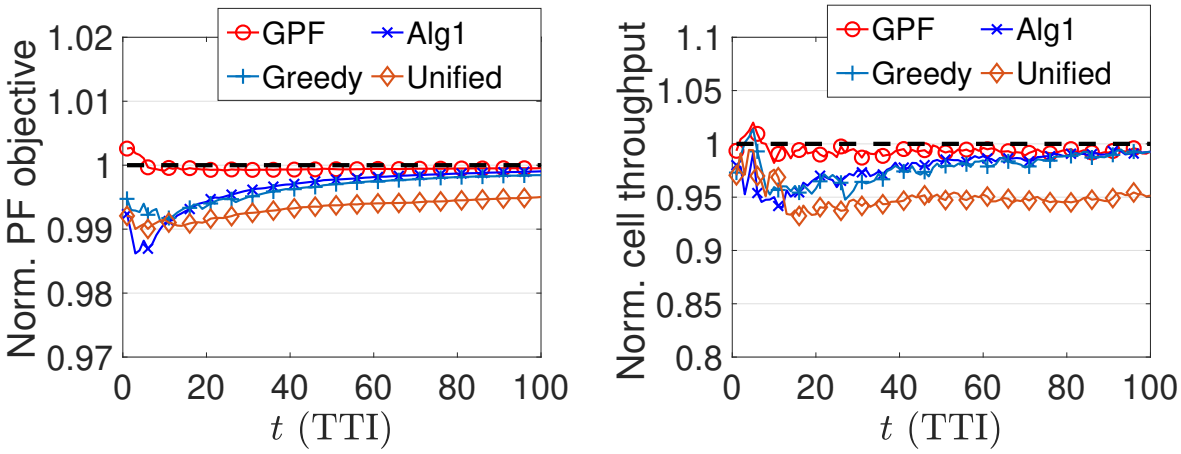
(a) Normalized PF objective

(b) Total cell throughput



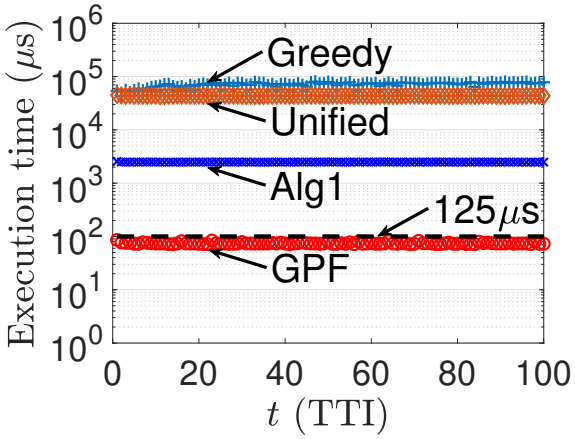
(c) Execution time

Figure 3.9: GPF+'s performance vs. other state-of-the-art PF schedulers for $U = 50$ users.



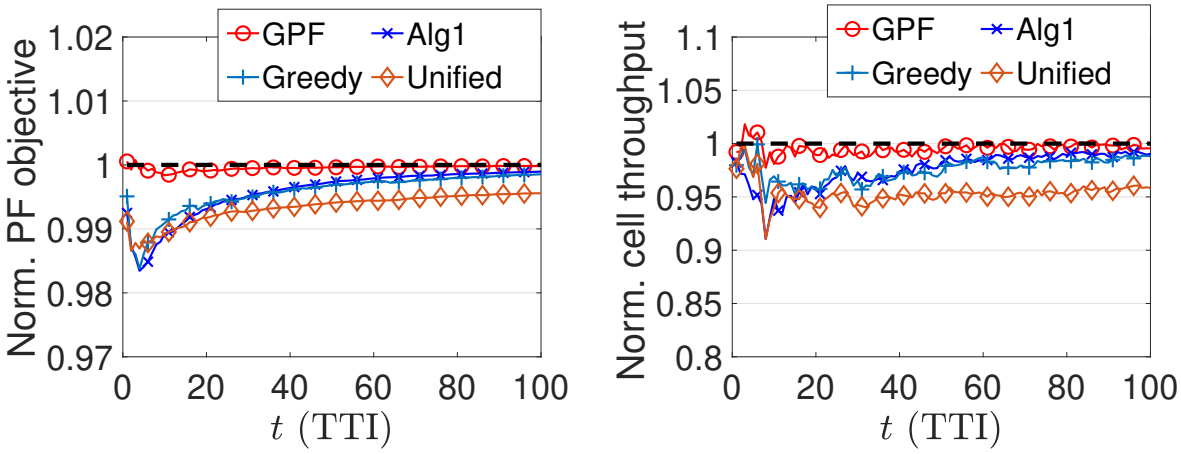
(a) Normalized PF objective

(b) Total cell throughput



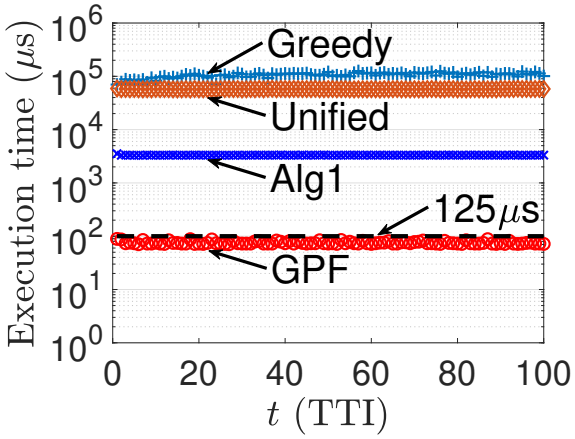
(c) Execution time

Figure 3.10: GPF+'s performance vs. other state-of-the-art PF schedulers for $U = 75$ users.



(a) Normalized PF objective

(b) Total cell throughput



(c) Execution time

Figure 3.11: GPF+'s performance vs. other state-of-the-art PF schedulers for $U = 100$ users.

Table 3.3: Breakdown of GPF's execution time consumed in different stages. Data format: (mean (μs), standard deviation).

	$U = 25$	$U = 50$	$U = 75$	$U = 100$
H-to-G	(14.27, 1.19)	(19.33, 0.95)	(24.01, 1.20)	(26.42, 1.52)
GPU	(30.32, 1.33)	(31.55, 1.18)	(36.75, 1.10)	(38.65, 1.04)
G-to-H	(10.59, 1.86)	(11.20, 3.84)	(13.03, 1.36)	(12.49, 1.09)
Total	(54.87, 2.02)	(61.49, 2.48)	(73.81, 3.08)	(76.17, 4.32)

the other PF schedulers under $U = 50$, 75 and 100, respectively. These results confirm that GPF+'s execution time is within 100 μs under all different user population sizes U's. The means and standard deviations of GPF+'s execution time are 61.72 μs and 2.64 for $U = 50$, 74.52 μs and 2.81 for $U = 75$, and 75.64 μs and 4.60 for $U = 100$, respectively. In contrast, *Alg1* (the fastest among the three PF schedulers for benchmarking) has a mean computation time of 1,658.23 μs, 2,487.07 μs, and 3,324.19 μs for U =50, 75, and 100, respectively. The average computation time of CPLEX is 4.73 s, 8.52 s, and 12.25 s for U =50, 75, and 100, respectively.

In conclusion, GPF+ is the only scheduler that can meet 125 μs timing requirement while achieving near-optimal PF performance.

Breakdown of Execution Time Next, we further investigate GPF+'s execution time spent at different stages, including the data transferring from host to GPU, the processing steps completed at GPU, and the solution transferring from GPU to host (see Section 3.7). Means and standard deviations of the execution time in each stage under different U's (each for 1,000 TTIs) are shown in Table 3.3. We find that the data transferring between host and GPU takes nearly half of the total execution time. The proportions of time spent on data transferring are 45.31%, 49.65%, 50.18% and 51.08% under U=25, 50, 75 and 100, respectively. Such time cost may be reduced by employing an integrated host-GPU architecture for algorithm implementation (e.g., see [79, 80]).

3.9 Chapter Summary

This chapter presents the first successful design of a PF scheduler that can meet the 100-μs time requirement in 5G NR. Our real-time design is based on a novel decomposition of the original optimization problem into a large number of small and independent sub-problems. By employing CE-based intensification and random sampling techniques, we are able to find a subset of promising sub-problems to fit in a given GPU platform for parallel processing. We implement our GPF+ on an off-the-shelf NVIDIA Tesla V100 GPU using the CUDA programming model. Through extensive experiments, we show that GPF+ can achieve near-optimal performance and meet the 100-μs time requirement while none of the existing state-of-the-art PF schedulers can do so. By analyzing the time overhead incurred at different stages across the host/GPU, we find that GPF+ has the potential to achieve even better timing performance if an integrated host/GPU system is employed.

To the best of our knowledge, at the time of submitting this article, our GPF+ design remains the world's fastest PF scheduler and is the only one available to meet 5G NR numerology 3 timing requirement. Although our GPF+ solution is developed to solve a PF scheduling problem, its underlying methodology is general and points to a new direction to address complex resource allocation problems in wireless networks.

Chapter 4

A DRL-based Approach to Dynamic eMBB/URLLC Multiplexing in 5G NR

4.1 Introduction

A major technical challenge in 5G NR is to support service types with extremely diverse performance requirements with a unified air interface [32]. Enhanced Mobile BroadBand (eMBB) and Ultra-Reliable and Low Latency Communications (URLLC) are two such important types of services in NR [34]. eMBB aims to offer a per-user data throughput higher than 100 Mb/s [81]. URLLC targets at mission-critical applications with low data rates (0.1-10 Mb/s) but extremely stringent latency requirements on $\sim$1 ms level [82]. To meet these service requirements, eMBB and URLLC employ very different time duration for data transmission. In NR, each time slot is divided into a number of mini-slots [35]. eMBB uses regular time slots for transmission to achieve high data throughput. On the other hand, URLLC data is transmitted in mini-slots so as to minimize the latency.

When eMBB and URLLC services are dynamically multiplexed on the same channel, the optimal allocation of radio resources becomes a challenging problem. To address this

problem, a novel resource allocation mechanism, termed "URLLC preemption", was proposed in 3GPP standards body [83]. The essence of this mechanism is to have an arriving URLLC packet preempt resources that have already been allocated to on-going eMBB transmissions. Specifically, when a URLLC packet arrives, its transmission is scheduled immediately for the next mini-slot. Depending on the packet size and the selected modulation and coding scheme (MCS), a URLLC packet requires a certain number of sub-carriers (SCs) to transmit. These SCs are obtained by directly puncturing the time-frequency resource grid in the scheduled mini-slot. The punctured SCs spread across resources allocated to different eMBB users. To keep track of (pinpoint) which resources have been taken by URLLC, special indication signals are generated and sent to eMBB users, which will be used for decoding. It has been shown that this preemptive puncturing approach is more spectrally efficient than static or semi-static spectrum separation between eMBB and URLLC, due to the sporadic and random nature of URLLC traffic [84].

Under this preemptive puncturing approach, an important optimization problem is the allocation of SCs preempted by each URLLC transmission among eMBB users. The decoding result of an eMBB user's received transmission directly depends on URLLC preemption: the more SCs being punctured by URLLC, the more likely the decoding will fail. Such decoding failure will lead to a loss of eMBB utility (e.g., sum of eMBB users' data rates or weighted rates). Therefore, the objective of our optimization problem is to minimize the loss of eMBB utility through properly spreading out preempted resources (for URLLC) among eMBB users in each transmission.

Unfortunately, the above optimization problem cannot be formulated and solved through traditional model-based optimization approaches. This is because the objective (cost) function of the problem requires the modeling of eMBB decoding behavior (e.g., failure probability) as a function of the amount of URLLC preemption in exact closed form. However, due

to complexities of NR PHY layer technologies such as LDPC channel code, it is impossible to obtain a closed-form expression for this cost function. As will be explained in Section 4.4, other approximate modeling or numerical approaches are also unsuitable for addressing this problem.

In this chapter, we present "DEMUX", a model-free deep (DE) reinforcement learning based multiplexer (MUX) for eMBB and URLLC services. The most significant advantage of DEMUX is that it offers a solution to the URLLC preemption problem without a prior explicit problem formulation. DEMUX uses deep function approximators (a.k.a., neural networks) to learn an optimal algorithm for determining preemption solution. Architecturally, DEMUX consists of a learning plane and a scheduling plane. The goal of the learning plane is to learn an optimal algorithm for URLLC preemption. The goal of the scheduling plane is to use the algorithm learned by learning plane to compute preemption solution for URLLC. Our main contributions in the design of DEMUX are summarized as follows:

- This work presents the first solution to the URLLC preemption problem that utilizes deep reinforcement learning (DRL). In the broader context, this is also the first work (to the best of our knowledge) that employs DRL method with large and continuous action domain to address resource scheduling problems in NR. This is significant as deep Q-learning methods (the most-commonly used learning methods for wireless communications) are only effective to address problems with discrete and small action space but are not suitable to solve problems with large and continuous action domain, such as the URLLC preemption problem that we study in this chapter.

- On the learning plane of DEMUX, although our design is inspired by the deep deterministic policy gradient (DDPG) method [85], we find that there exists serious convergence issue with this approach in addressing our problem. To mitigate this issue,

we propose to augment DDPG method by exploiting some intrinsic properties of the URLLC preemption problem. This includes adapting the learning objective of DDPG based on scheduling mechanism of eMBB and eliminating the use of target networks in DDPG without hampering the stability of the learning process.

- On the scheduling plane, a major problem is that the learned algorithm, which is an unconstrained neural network, cannot guarantee to offer a feasible solution for URLLC preemption. To address this problem, we propose a novel approach based on the concept of relative entropy in information theory to translating the raw output of the learned algorithm (neural network) into a feasible preemption solution. We show that our proposed solution is optimal in terms of minimizing the loss of information caused by such translation.

- In our implementation, we build a PHY-MAC NR simulator for dynamic eMBB/URLLC multiplexing and a learning module based on our design of DEMUX. Our implementation captures essential functions in PHY and MAC layers of NR that are related to URLLC preemption as well as interaction mechanisms between an NR base station (BS) and a DRL learning module. Our NR simulator is tuned and validated through extensive experiments to ensure correctness of the PHY- and MAC-layer functions (e.g., the measurement of working SNRs to ensure target BLERs for different MCS levels, validation for soft demodulation and decoding under URLLC preemption, among others.)

- Using our NR simulator, we validate the performance of DEMUX through benchmarking with two other preemption schemes proposed in the literature (RP from [86] and FFP from 3GPP standards body [87, 88]) under different multi-path channel fading models. These benchmark schemes were developed based on various modeling assumptions and thus are considered as heuristic solutions to the URLLC preemption problem.

Experiment results show that DEMUX significantly outperforms RP and FFP in terms of both sum utility and throughput in a cell. In particular, when URLLC traffic load is high, DEMUX is able to achieve more than 130% and 75% performance gains over RP and FFP respectively in our test network scenarios.

The rest of the chapter is organized as follows. In Section 4.2, we provide necessary background on dynamic eMBB/URLLC multiplexing and introduce the optimal URLLC preemption problem. In Section 4.3, we review existing works on the URLLC preemption problem in the literature. In Section 4.4, we go deep into technical challenges in solving the problem and show why model-based optimization approaches cannot be applied. In Section 4.5, we give an overview of DEMUX - our proposed solution to the URLLC preemption problem. In Section 4.6, we present DEMUX's scheduling plane, while in Section 4.7 we present DEMUX's learning plane. Our implementation of DEMUX is presented in Section 4.8. Section 4.9 presents experimental results based on our implementation. Section 4.10 concludes this chapter.

4.2 Dynamic Multiplexing of eMBB and URLLC

When eMBB and URLLC services are multiplexed on the same downlink channel, the allocation of time-frequency resources becomes very complicated. This is because eMBB and URLLC transmissions have different duration and are scheduled on very different time scales. In NR, the time domain of a channel is divided into time slots, with each time slot being further divided into multiple mini-slots. The duration of an eMBB transmission, termed a *transmission time interval* (TTI), may span one or multiple time slots. In contrast, each URLLC packet is transmitted using a mini-slot.

URLLC Preemptive Puncturing In this chapter, we investigate "URLLC resource pre-

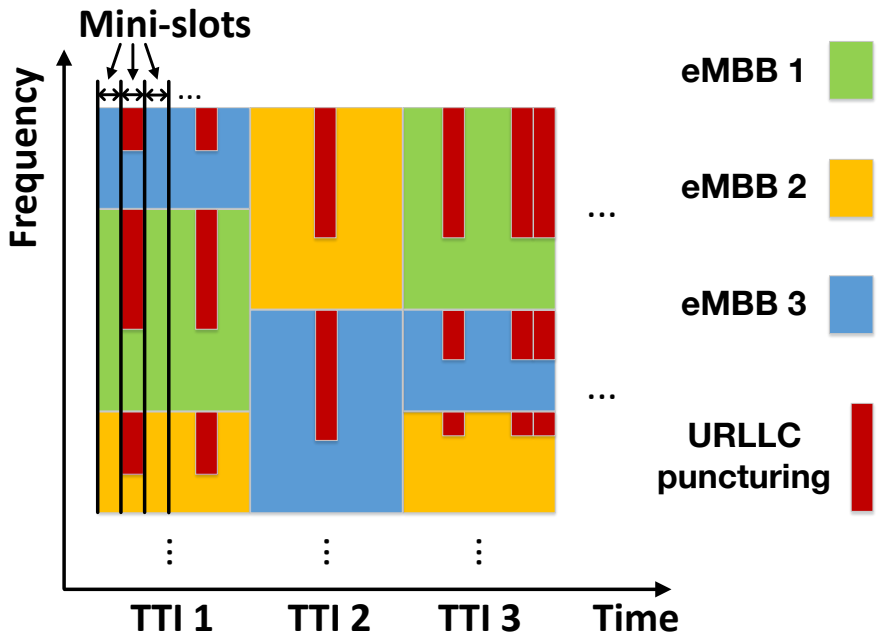


Figure 4.1: An illustration of URLLC resource preemption.

emption," a mechanism for dynamic multiplexing of eMBB and URLLC that was proposed in 3GPP standards body [83]. Fig. 4.1 illustrates this mechanism. The frequency domain of a channel consists of a large number of contiguous SCs. In each TTI, all SCs on the channel are allocated to eMBB users using some scheduling algorithm (e.g., proportional-fairness (PF) [52, 55]). On the other hand, URLLC packet arrivals are sporadic and random in time. When a URLLC packet arrives, its transmission is scheduled immediately for the next mini-slot.[1] Each URLLC packet requires a certain number of SCs to transmit (smaller than the total number of SCs on the channel). As all SCs have already been allocated to eMBB users, an incoming URLLC packet obtains its required SCs by puncturing/preempting resources from on-going eMBB transmissions. That is, a portion of SCs allocated to each eMBB user are reassigned to URLLC in the scheduled mini-slot. As shown in Fig. 4.1, the preempted SCs do not need to be contiguous in frequency [86]. Then data bits of eMBB are flushed

[1]In this chapter, we assume that at most one URLLC packet can be transmitted in a mini-slot. This is because a URLLC transmission generally requires very broad frequency bandwidth (may be more than half of the entire channel bandwidth).

out on those SCs preempted by URLLC.

In the example shown in Fig. 4.1, there are three, two, and three eMBB transmissions in TTI 1, 2, and 3, respectively. The numbers of URLLC transmissions in TTI 1, 2, and 3 are two, one, and three, respectively. Specifically, in TTI 1, there is a URLLC transmission in the second mini-slot and another one in the fifth mini-slot. Note that the SCs punctured by each URLLC packet are spread out across multiple eMBB users.

Under this preemption mechanism, eMBB code word bits are corrupted on SCs punctured by URLLC. To help eMBB users decode, the BS sends an indication signal to each eMBB user at the end of a TTI to specify positions of preempted resources in time and frequency [89, 91]. Based on this indication, eMBB users can precisely eliminate the corrupted bits before LDPC decoding. This will avoid polluting other code word bits during decoding and improve the decoding success probability.

Problem Statement For each TTI, eMBB transmissions are scheduled with a specific objective, e.g., maximizing the sum utility of eMBB users (such as the sum cell throughput or PF utility). The actually achieved eMBB utility, however, depends on whether or not each eMBB user can successfully decode its received transmission. If all transmissions are successful, then the maximum utility is preserved, while failed transmissions result in a loss of utility.

As expected, the decoding failure probability of an eMBB transmission increases with the amount of URLLC preemption. Since each URLLC packet can preempt SCs from multiple eMBB users, an important optimization problem to solve is: *For each URLLC transmission, how many SCs should we preempt from each eMBB user so that the loss of sum eMBB utility is minimized?* This is the problem that we will investigate in this chapter.

For each eMBB user, we assume that a contiguous block of SCs starting from the low-

est/highest frequency will be preempted (when preemption happens). We will not further optimize positions of preempted SCs within each eMBB user's resources since such an optimization will incur excessive overhead for feedback and control signaling. A practical set of constraints for URLLC puncturing are that the number of SCs preempted from each eMBB user cannot exceed the number of SCs allocated to her. Such constraints must be satisfied by any feasible preemption solution for URLLC transmissions.

4.3 Related Work

There have been a number of previous works on dynamic multiplexing of eMBB and URLLC based on preemptive puncturing [86, 87, 88, 92]. In 3GPP standards body [87, 88], a very simple (and easy to implement) preemption scheme was proposed. We call this scheme fixed-frequency-part (FFP) in this chapter. FFP statically assigns a contiguous block of SCs on the channel for each URLLC transmission. As we will show in Section 4.9, such a static allocation (preemption) scheme achieves poor performance in terms of maximizing eMBB utility. This demonstrates the critical need for optimizing URLLC preemption among eMBB users, which is the problem that we study in this chapter.

The works in [86] and [92] explored optimizations of URLLC preemption based on assumptions of specific mathematical models for the cost function. Here, the cost function represents the loss of eMBB utility as a function of the amount of resources (SCs) preempted from each eMBB user. Under this approach, in [86], the authors assumed different cost function models (linear or nonlinear) and developed a customized solution for each model. Specifically, under a linear cost function model, a solution that preempts SCs in proportion to each eMBB user's resource allocation was found to be optimal. Under a nonlinear cost function model, a non-closed-form solution based on standard optimization formulation was

proposed. In [92], the authors assumed a linear model for cost function and formulated an optimization problem for resource allocation between eMBB and URLLC. Then a matching based algorithm was proposed to solve the formulated problem.

A recent work [93] studied machine learning based resource scheduling for eMBB/URLLC multiplexing on the same channel. The flexible setting of TTI duration for different services was considered as an enhancement to traditional fixed-length TTI based scheduling. However, the multiplexing mechanism considered in [93] is fundamentally different from the URLLC puncturing scheme that we study in this chapter. The proposed random forest based classification algorithm cannot be easily extended to address our URLLC preemption optimization problem.

4.4 Why Model-based Approaches Will Not Work

Although a model-based approach is attractive from a mathematical perspective, it suffers from the following major flaw. Specifically, the underlying assumption that the optimal URLLC preemption problem can be abstracted and modeled with a mathematical formulation does not hold. The main reason here is that the cost (objective) function of the problem cannot be modeled in closed form. Inputs to this cost function include URLLC preemption among eMBB users, MCS selection and SC allocation for each eMBB user, channel conditions across SCs, among others. As we shall discuss in the rest of this section, it is mathematically intractable to express this cost function through either exact analytical approaches, simplified approximations, or numerical methods.

First, the cost function cannot be obtained through analytical abstraction and modeling due to the complexities involved in NR PHY layer. In NR, LDPC is employed as the forward error correction (FEC) channel code for data transmission [91, 94]. It is well known that

LDPC decoding performance is very difficult to analyze and cannot be modeled exactly. There is no closed-form expression for the error-correction capability of LDPC.[2] As a result, we cannot obtain an explicit expression for the block error rate (BLER) performance of eMBB. In practice, the BLER curves can only be determined numerically through extensive experiments or simulations [91].

Second, given that an exact model for the cost function is not obtainable, one might attempt to develop a simplified approximate model and use it for analysis and optimization. Unfortunately, such an approach is unlikely to be successful because simple approximations are unable to accurately characterize the complex impact of URLLC preemption on eMBB performance. As an example, in [86], a linear model was assumed for the cost function and subsequently a so-called *Resource Proportional (RP) Placement* algorithm was proposed and found to be optimal under this linear cost function. But its actual performance under practical system settings is rather poor (see Section 4.9), due to the inaccuracy of a linear model.

Finally, one might wonder if it is possible to estimate the cost function numerically by enumerating all possible inputs, which include, among others, the amount of URLLC preemption, MCS and SC assignment for each eMBB user, and channel conditions across SCs. However, the input space of this approach is prohibitively large, rendering this numerical approach practically infeasible.

[2]Refer to [90] for a state-of-the-art lower bound for the error-correction capability of LDPC, which is not in closed form.

4.5 DEMUX: An Overview

In light of discussions in the last section, it is clear that a very different approach is needed to address the URLLC preemption problem. In this chapter, we propose DEMUX – a novel model-free DRL-based solution to this problem. In this section, we offer an overview for the overall architecture of DEMUX. More design details will be presented in Section 4.6 and 4.7.

4.5.1 Why DRL

Model-free DRL is a machine learning technique that employs deep function approximators (neural networks) to learn an *unknown* optimal solution algorithm without a prior problem modeling. For addressing our URLLC preemption problem, the most attractive feature of model-free DRL is that it does not require an explicit cost function formulation. In addition, DRL is particularly suitable for solving problems with very large input spaces (e.g., the space of all possible inputs for our problem). An algorithm (a neural network) learned properly through DRL can perform well for input instances that have never been tried in the learning phase (the so-called *generalization capability* of neural networks [95]). Thus it is possible to learn an optimal algorithm from a relatively small subset of input instances. Finally, DRL is extremely versatile and can learn a wide range of complex algorithms with suitable neural network structures [95]. For example, by adding more fully-connected layers to a neural network, one can generally enhance its capability of representing more complex functions. But with more layers, a neural network will have longer forward propagation time. When addressing problems with real-time requirements (as our problem), there is a design trade-off between the function representation capability of a neural network and its computational time cost.

Table 4.1: Notation in Chapter 4

Symbol	Definition
C_n	eMBB user n's long-term average data rate
f	The translator function in the scheduling plane
g	The post-processor function in the scheduling plane
K	The mini-batch size for random sampling in replay buffer
L_{URLLC}	The number of SCs required for each URLLC transmission
M^{SC}	The total number of SCs on the operating channel
M^{RBG}	The total number of RBGs on the operating channel
Q	The critic neural network
$r(t)$	The reward received for TTI t
$\boldsymbol{s}(t)$	The eMBB instance in TTI t
$U(t)$	The achieved eMBB utility for TTI t
$\boldsymbol{\alpha}(t)$	The output of μ in TTI t
$\tilde{\boldsymbol{\alpha}}(t)$	The output of g in TTI t
$\boldsymbol{\beta}(t)$	The URLLC puncturing solution for TTI t
$\boldsymbol{\gamma}(t)$	The eMBB resource allocation vector for TTI t
$\boldsymbol{\delta}(t)$	The eMBB MCS selection vector for TTI t
$\boldsymbol{\epsilon}(t)$	The indication vector for eMBB transmission failures in TTI t
Φ	The set of trainable parameters in Q
Θ	The set of trainable parameters in μ
λ	URLLC arrival rate (in number of packets/TTI)
π	An (unknown) optimal algorithm for URLLC puncturing
μ	The deep neural network used to approximate π
$\bar{\pi}$	$= f \circ g \circ \mu$, the overall algorithm of the scheduling plane

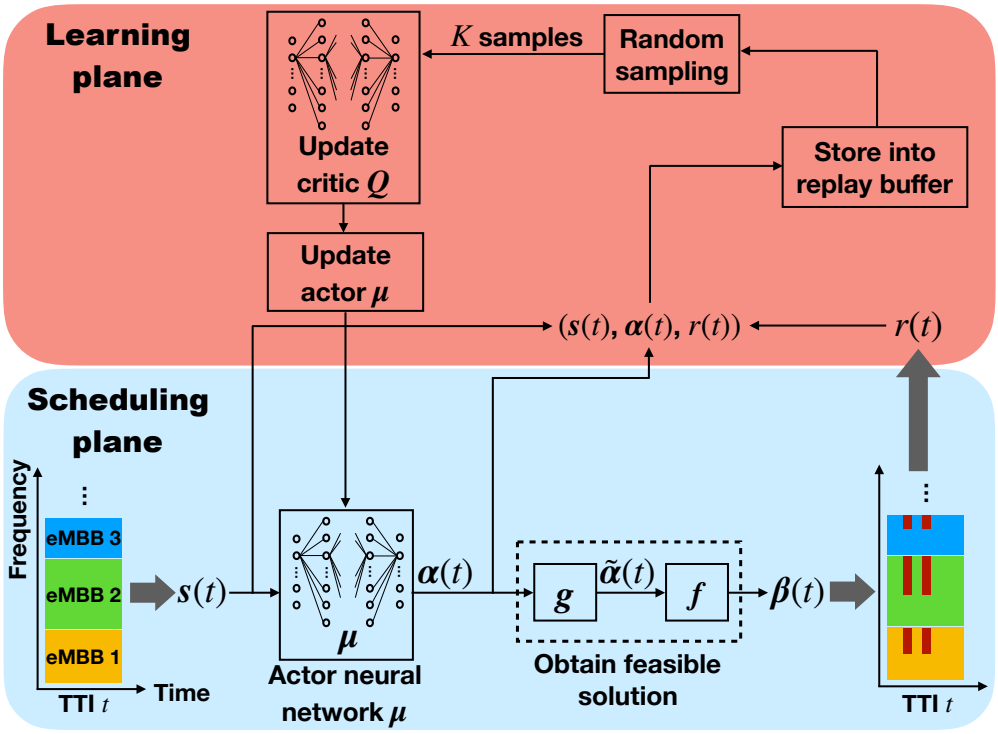


Figure 4.2: Scheduling plane and learning plane in DEMUX.

4.5.2 Architecture of DEMUX

At the beginning of each TTI, we have no knowledge whether or not and how many URLLC packets will arrive among the mini-slots in this TTI. The only available information is URLLC's estimated arrival rate (based on online measurements). With this limited information, the best thing we can do is to determine a preemption solution based on the estimated URLLC arrival rate, which will be used for all URLLC packets that will actually arrive in this TTI. This preemption solution specifies how many SCs should be preempted in a mini-slot from each eMBB user for a URLLC transmission (see Fig. 4.1). Therefore, at the beginning of each mini-slot, if a URLLC packet is present, then its required SCs are obtained by puncturing resources from eMBB users following the preemption solution for this TTI. On the other hand, if there is no URLLC transmission in a mini-slot, then no preemption will occur.

The goal of DEMUX is, therefore, to obtain an optimal algorithm to compute a URLLC preemption solution for each TTI. Suppose there exists an *unknown* optimal algorithm for URLLC preemption, denoted by π, that can minimize the loss of eMBB utility in each TTI (under a given URLLC arrival rate). Then our goal for DEMUX is to find an approximation μ of this optimal algorithm π using a DRL-based approach. In particular, μ is a deep function approximator (neural network) with a large number of trainable parameters within its structure. In fact, since the URLLC packet arrival rate may change over time, we will learn a different algorithm μ for each different (sampled) URLLC arrival rate (see Section 4.8.3).

Table 4.1 lists notation used in this chapter. The architecture of DEMUX consists of a *scheduling plane* and a *learning plane,* similar to the data plane and the control plane in computer networks. Figure 4.2 illustrates our algorithms on both planes. The role of learning plane is to optimize parameters in μ through a large number of learning steps. The role of scheduling plane is to use algorithm μ for real-time multiplexing of eMBB and URLLC traffic. In the learning phase, the two planes work in a closed loop through a series of procedures to optimize μ so that it can approximate an optimal algorithm π for URLLC preemption. The controller of the whole learning process resides at the BS. The eMBB users only need to report their decoding results (success or failure) to the BS, which will then use them to construct a reward function.

Scheduling Plane The input to the scheduling plane is a vector $\boldsymbol{s}(t)$. In particular, $\boldsymbol{s}(t)$ contains the resource allocation and MCS selection information for each eMBB user. In each TTI t, $\boldsymbol{s}(t)$ is first fed into the neural network μ, which then produces an output vector $\boldsymbol{\alpha}(t)$. Since $\boldsymbol{\alpha}(t)$ is the unconstrained output of a neural network, it is not guaranteed to be a *feasible* solution for URLLC preemption. To address this issue, we design a post-processor g and a translator f that can obtain a feasible preemption solution $\boldsymbol{\beta}(t)$ based on $\boldsymbol{\alpha}(t)$. The purpose of g is to refine $\boldsymbol{\alpha}(t)$ so that its output $\tilde{\boldsymbol{\alpha}}(t)$ will match the eMBB resource allocation

structure as $\boldsymbol{s}(t)$. But $\tilde{\boldsymbol{\alpha}}(t)$ may still violate the constraints that URLLC preemption cannot exceed resource allocation for each eMBB user. Next, we use f to convert $\tilde{\boldsymbol{\alpha}}(t)$ into a feasible solution $\boldsymbol{\beta}(t)$ with a minimum loss of information (based on KL divergence). Entries of $\boldsymbol{\beta}(t)$ indicate the proportion of SCs to be preempted from each eMBB user. Finally, $\boldsymbol{\beta}(t)$ will be used for all URLLC transmissions scheduled in mini-slots within TTI t.

Denote $\bar{\pi}$ as the overall algorithm on the scheduling plane (which takes $\boldsymbol{s}(t)$ as input and outputs preemption solution $\boldsymbol{\beta}(t)$). Then we have $\bar{\pi} = f \circ g \circ \mu$, where $\circ$ represents composition of functions. Details for the design of $\bar{\pi}$ will be presented in Section 4.6. As a performance measure of $\bar{\pi}$, a reward function $r(t)$ is generated and sent to the learning plane.

An important requirement for the scheduling plane is that the total processing time of $\bar{\pi}$ must be less than the duration of a TTI. This is to ensure that preemption solution $\boldsymbol{\beta}(t)$ can be updated in each TTI in a real-time manner. For example, when Numerology 1 (500 μs time slot duration) is used for the channel, the duration of a single-slot TTI is 0.5 ms. Then the aggregate processing time of μ, g, and f must be no more than 0.5 ms. This real-time requirement is one of our key design targets for the scheduling plane.

Learning Plane The learning plane is inspired by the DDPG method [85] but with our own contributions to address the URLLC preemption problem. A fundamental issue with using DDPG for our problem is that the learning process is extremely slow (i.e., not converging even over half a million iterations). To tackle this issue, we adapt the learning objective of DDPG based on intrinsic properties of the problem and cut down the number of deep function approximators in DDPG from four to two, i.e., only having μ and Q, where Q is a critic neural network. In Section 4.7.4, we will show that the proposed learning method is effective for our objective (i.e., minimizing the loss of eMBB utility in each TTI).

Our learning method has an actor-critic structure similar to DDPG (more details in Section 4.7.2). The learning of approximate algorithm μ is done with a large number of iterations. In each iteration, a 3-tuple instance $(\boldsymbol{s}(t), \boldsymbol{\alpha}(t), r(t))$ is first received from scheduling plane. This 3-tuple instance is stored in a replay buffer of finite capacity. When the buffer is full, the oldest instance in it is pushed out (discarded). Then a subset of K random samples $(\boldsymbol{s}(i), \boldsymbol{\alpha}(i), r(i)), i = 1, \cdots, K$ are taken from the replay buffer. These samples are first used for updating parameters in critic neural network Q. As a key component of learning plane, Q is used for predicting the expected reward $\hat{r}(t)$ for an instance $(\boldsymbol{s}(t), \boldsymbol{\alpha}(t))$, i.e., the reward one would expect to receive when μ outputs $\boldsymbol{\alpha}(t)$ with input eMBB instance $\boldsymbol{s}(t)$. Parameters in Q are updated by minimizing $\sum_{i=1}^{K}(r(i) - \hat{r}(i))^2$. After updating Q, the K samples are further used for updating parameters in algorithm μ. A gradient of Q w.r.t. parameters in μ is computed with the K samples. Then parameters in μ are updated along this gradient. After these updates, algorithm μ is sent to the scheduling plane for multiplexing of eMBB and URLLC.

4.6 Design of the Scheduling Plane

The overall algorithm on the scheduling plane is $\bar{\pi} = f \circ g \circ \mu$. This section presents how each of these three components of $\bar{\pi}$ is designed. To ease our notation, we omit t after TTI when there is no confusion.

4.6.1 An Example

To help us understand how $\bar{\pi}$ works in each TTI, consider the following illustrative example (not entirely conforming to NR standard). Suppose there is a total of 100 SCs on the channel

and the minimum resolution for resource allocation to eMBB is 20 SCs. This means that an eMBB user, if chosen for SC allocation, can be allocated with either 20, 40, 60, 80, or 100 SCs. It also suggests that there are at most five eMBB users that can be chosen (from the total user pool) for SC allocation and transmission per TTI.

Suppose at the beginning of a TTI, we observe an eMBB instance that is characterized by the following vector:

$$\boldsymbol{s} = \big[\underbrace{0.2 \;\; 0.4 \;\; 0.2 \;\; 0.2 \;\; \text{null}}_{\text{SC allocation}} \;\; \underbrace{0.21 \;\; 0.58 \;\; 0.10 \;\; 0.39 \;\; \text{null}}_{\text{MCS selection}}\big],$$

where the left five elements inside the brackets represent SC allocation for scheduled eMBB users (up to five) and the right five elements represent MCS selection for each eMBB user (in terms of code rates).

The eMBB instance $\boldsymbol{s}$ is fed into the deep function approximator μ. Then μ outputs an unconstrained preemption instruction $\boldsymbol{\alpha}$ as follows:

$$\boldsymbol{\alpha} = [0.35 \;\; 0.11 \;\; 0.48 \;\; 0 \;\; 0.06].$$

Each element in $\boldsymbol{\alpha}$ represents the proportion of SCs (required by a URLLC packet) that should be preempted from an eMBB user. All elements in $\boldsymbol{\alpha}$ add up to one.

Apparently $\boldsymbol{\alpha}$ is not a feasible solution, as the fifth element of $\boldsymbol{\alpha}$ should be zero since there are only 4 eMBB transmissions. To make this correction, $\boldsymbol{\alpha}$ is sent to a post-processor g. g first zeros out redundant entry in $\boldsymbol{\alpha}$ (the fifth entry). Then a small random positive disturbance (0.02 in this example) is added to the fourth zero entry. This operation is required by the translator f (will be explained in Section 4.6.3). Finally, g normalizes the

sum of all elements to 1. The output of g, denoted as $\tilde{\boldsymbol{\alpha}}$, is:

$$\tilde{\boldsymbol{\alpha}} = [0.36 \;\; 0.12 \;\; 0.50 \;\; 0.02 \;\; \text{null}].$$

But $\tilde{\boldsymbol{\alpha}}$ still may not meet feasibility constraints for URLLC preemption. For example, suppose the current URLLC transmission requires 50 SCs. The third element of $\tilde{\boldsymbol{\alpha}}$ says that 50% of the 50 SCs (=25 SCs) should be preempted from eMBB user 3. But eMBB user 3 is only allocated with 20 SCs ($= 100 \times 20\%$). To address this infeasibility, $\tilde{\boldsymbol{\alpha}}$ is sent to a translator f, which will produce a feasible preemption solution $\boldsymbol{\beta}$ as follows:

$$\boldsymbol{\beta} = [0.43 \;\; 0.15 \;\; 0.40 \;\; 0.02 \;\; \text{null}].$$

Then $\boldsymbol{\beta}$ will be used for all URLLC transmissions in this TTI.

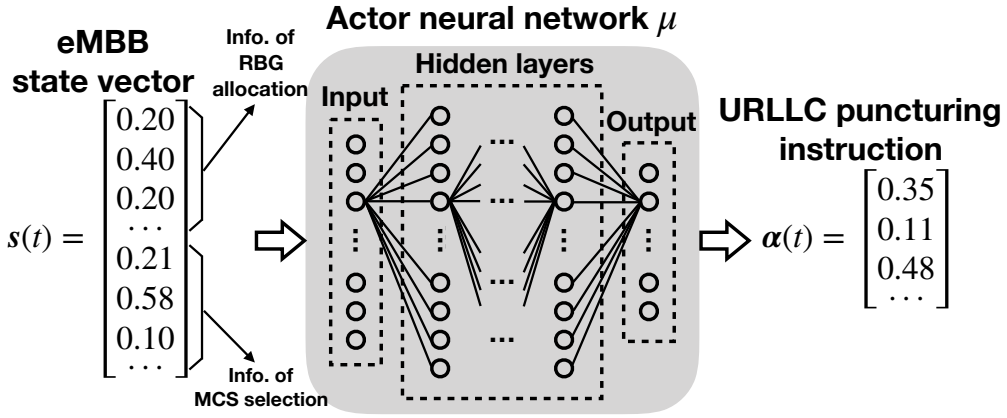
In the rest of this section, we elaborate the design details of μ, g, and f.

4.6.2 Approximate Algorithm μ

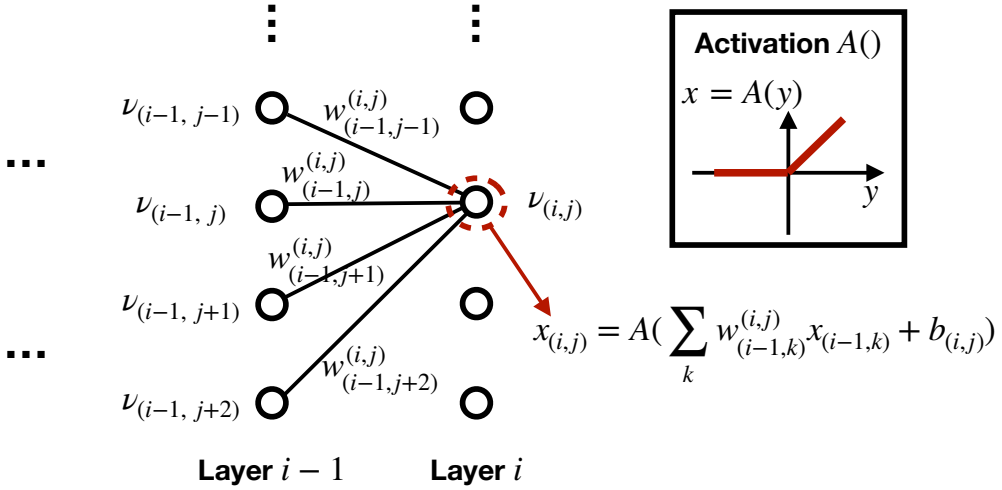
The approximate algorithm μ in the scheduling plane is a neural network consisting of an input layer, a number of intermediate layers and an output layer. An illustration of μ is given in Fig. 4.3. μ contains a large set of trainable parameters, which we denote as Θ. These parameters are optimized by the learning plane as illustrated in Fig. 4.2.

Input and Output At the beginning of each TTI, the input to μ is an eMBB instance vector $\boldsymbol{s}$, as described in the example in Section 4.6.1. In general, $\boldsymbol{s}$ can be expressed as:

$$\boldsymbol{s} = [\boldsymbol{\gamma} \; \boldsymbol{\delta}], \tag{4.1}$$



(a) Layered structure of μ.



(b) Forward propagation between two adjacent layers.

Figure 4.3: Deep function approximator μ.

where $\boldsymbol{\gamma}$ is the normalized SC allocation vector (i.e., percentage of SCs allocated to each eMBB user) and $\boldsymbol{\delta}$ is the MCS selection vector (i.e., MCS selection for each eMBB user). In NR, a resource block group (RBG) is the minimum frequency resolution for eMBB resource allocation [36]. Each RBG consists of a number of contiguous resource blocks (RBs), while an RB contains 12 contiguous SCs. Let M^{RBG} denote the total number of RBGs on the channel. In each TTI, an RBG can be allocated to at most one eMBB user. Thus the maximum number of eMBB users that can be scheduled per TTI is M^{RBG}. For the example in Section 4.6.1, there are $M^{\text{RBG}} = 5$ RBGs on the channel with each RBG containing 20 SCs. To ensure a uniform input format for each TTI, we fix the lengths of both $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ to M^{RBG}. Thus the size of the input vector $\boldsymbol{s}$ is $2M^{\text{RBG}}$. When the number of scheduled eMBB users is less than M^{RBG}, we can pad the unspecified elements in $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ to null (zero).

The output of μ is a vector $\boldsymbol{\alpha}$ with M^{RBG} elements. The softmax function is used before output layer $\boldsymbol{\alpha}$. Then each element of $\boldsymbol{\alpha}$ is a real number between 0 and 1, representing the proportion of SCs required by a URLLC transmission that should be preempted from the corresponding eMBB user. Denote L_{URLLC} as the number of SCs required by a URLLC transmission.[3] The sum of entries in $\boldsymbol{\alpha}$ equals to 1, meaning that the number of SCs preempted from all eMBB users equals to L_{URLLC}.

Intermediate Layers Between the input and output layers of μ, we have a multi-layer structure that is fully connected between two adjacent layers. Such a structure can offer a strong function approximating capability, which is what we need for μ to approximate optimal algorithm π. Under this structure, each intermediate layer consists of a number of "neurons". A neuron within a layer is fully connected to all neurons in the next layer (as shown in Fig. 4.3a). Forward propagation from input to output between two adjacent

[3] L_{URLLC} is determined by the packet size (in number of bits), the MCS used for URLLC, and the number of OFDM symbols per mini-slot. For example, for a 50-byte packet size, QPSK modulation, LDPC with 1/3 code rate, and 2 OFDM symbols per mini-slot, we have $L_{\text{URLLC}} = 300$.

layers is shown in Fig. 4.3b. For example, consider the j-th neuron in the i-th layer of μ, which we denote as $\nu_{(i,j)}$. The input to $\nu_{(i,j)}$ is the weighted sum of all neurons' outputs in the $(i-1)$-th layer: $\sum_k w^{(i,j)}_{(i-1,k)} \cdot x_{(i-1,k)}$, where $x_{(i-1,k)}$ is the output of $\nu_{(i-1,k)}$ and $w^{(i,j)}_{(i-1,k)}$ is the weight on the connection from $\nu_{(i-1,k)}$ to $\nu_{(i,j)}$. Then the output of $\nu_{(i,j)}$ is: $x_{(i,j)} = A\left(\sum_k w^{(i,j)}_{(i-1,k)} \cdot x_{(i-1,k)} + b_{(i,j)}\right)$, where $b_{(i,j)}$ is the bias of $\nu_{(i,j)}$ and $A(\cdot)$ denotes the non-linear activation function, e.g., rectified non-linearity. Weights w's and biases b's are contained in Θ and optimized by the learning plane.

Under the real-time requirement in NR, the aggregate processing time of μ, g, and f must be no more than the duration of a TTI. To meet this requirement, we must avoid using too many intermediate layers in μ since forward propagation of μ increases with the number of layers. In our implementation of DEMUX in Section 4.8, in order to satisfy the requirement for NR Numerology 0 and 1 (with slot duration of 1 ms and 0.5 ms, respectively, most suitable for eMBB), a four-layer structure (with two intermediate layers) is used for μ to achieve a forward propagation time of 0.276 ms with an Intel Xeon E5-2687W v4 CPU. For NR Numerology 2 and 3 (with slot duration of 0.25 ms and 0.125 ms, respectively), additional mechanism (e.g., GPU [96]) for speedup would be necessary.

4.6.3 Guaranteeing Feasibility with g and f

Infeasibility An issue with approximate algorithm μ is that its output $\boldsymbol{\alpha}$ is unconstrained and may not be a feasible solution for URLLC preemption (as shown in the example in Section 4.6.1). Such infeasibility may come from two sources:

(i) $\boldsymbol{\alpha}$ may have non-zero elements corresponding to null elements in $\boldsymbol{\gamma}$;

(ii) $\boldsymbol{\alpha}$ may ask for more SCs to preempt than what the corresponding eMBB users' SCs can offer.

Next, we present a post-processor g and a translator f to address these potential infeasibility issues, respectively.

Post-processor g The goal for post-processor g is to address the first infeasibility issue. To do this, g performs the following operations on $\boldsymbol{\alpha}$. First, we use a mask vector of size M^{RBG} that has the same number of null entries as $\boldsymbol{\gamma}$ and ones elsewhere. We use this mask vector to nullify those "infeasible" entries in $\boldsymbol{\alpha}$.

Also, if there is any entry in $\boldsymbol{\alpha}$ that equals to zero while the same entry in $\boldsymbol{\gamma}$ is nonzero, we will add a very small positive disturbance to this entry in $\boldsymbol{\alpha}$. This operation has minimal impact on the final integral preemption solution but is required by translator f (to be explained later). Finally, we normalize the sum of non-zero elements from the previous steps in g to 1. Through the processing of g, we obtain a new preemption instruction $\tilde{\boldsymbol{\alpha}}$.

Translator f But $\tilde{\boldsymbol{\alpha}}$ may still be infeasible, due to the second infeasibility issue. The goal of translator f is to address the second infeasibility so that we can get a final feasible preemption solution $\boldsymbol{\beta}$. Given a resource allocation vector $\boldsymbol{\gamma}$ for eMBB users, a feasible $\boldsymbol{\beta}$ should ensure that the number of SCs preempted from each eMBB user is no greater than the total number of SCs allocated to this user. That is, $\boldsymbol{\beta}$ must satisfy $L_{\text{URLLC}}\beta_n \leq M^{\text{SC}}\gamma_n$ for each scheduled eMBB user n, where M^{SC} is the total number of SCs on the channel, and β_n and γ_n are the n-th entries of $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$, respectively. Then we have the following feasibility constraints:

$$\beta_n \leq \gamma_n \frac{M^{\text{SC}}}{L_{\text{URLLC}}} \quad n = 1, 2, \cdots, M^{\text{RBG}}. \tag{4.2}$$

Per our discussion about the second infeasibility issue, $\tilde{\boldsymbol{\alpha}}$ (output of g) may not satisfy constraints (4.2). That is, there might be some entry $\tilde{\alpha}_n$ with $\tilde{\alpha}_n > \gamma_n \frac{M^{\text{SC}}}{L_{\text{URLLC}}}$. In fact, by experiments we found that such a violation is very common. To address this infeasibility, we propose a novel translator function f as follows.

Consider an approach where SCs are preempted from eMBB users randomly using $\boldsymbol{\beta}$ as a probability distribution (instead of fixed percentages). It is easy to see that in terms of expected number of SCs preempted from each eMBB user, this probabilistic approach and fixed percentage approach are equivalent. Therefore, we can view $\tilde{\boldsymbol{\alpha}}$ as a probability distribution for random preemption. Following this approach (i.e., treating $\tilde{\boldsymbol{\alpha}}$ and $\boldsymbol{\beta}$ as two probability distributions), a plausible way to construct a feasible $\boldsymbol{\beta}$ from the infeasible $\tilde{\boldsymbol{\alpha}}$ is to minimize the *distance* between the two. In this regard, we can use the Kullback-Leibler (KL) divergence as the measure of distance between the two probability distributions [97] and minimize this distance.

Denote $D^{\mathrm{KL}}(\cdot \parallel \cdot)$ as the KL divergence between two probability distributions. Then the KL divergence between $\tilde{\boldsymbol{\alpha}}$ and $\boldsymbol{\beta}$ can be defined by[4]

$$D^{\mathrm{KL}}(\tilde{\boldsymbol{\alpha}} \parallel \boldsymbol{\beta}) = \sum_{n} \tilde{\alpha}_n \log \frac{\tilde{\alpha}_n}{\beta_n}. \tag{4.3}$$

In information theory, KL divergence is also called relative entropy and is widely used for quantifying the loss of information from one probability distribution to the other probability distribution [98, 99]. Here, we will use (4.3) to measure how much information is lost if we use $\boldsymbol{\beta}$ to approximate $\tilde{\boldsymbol{\alpha}}$.

It is easy to show that $D^{\mathrm{KL}}(\tilde{\boldsymbol{\alpha}} \parallel \boldsymbol{\beta}) \geq 0$. Then the role of translator f is to find $\boldsymbol{\beta}$ that minimizes the KL divergence w.r.t. $\tilde{\boldsymbol{\alpha}}$ while meeting the feasibility constraints (4.2). Denote $\boldsymbol{\beta}^* = \{\beta_1^*, \beta_2^*, \cdots\}$ as the optimal solution found by f. To determine $\boldsymbol{\beta}^*$, f performs the following steps:

(i) Denote $\mathcal{N}^0 = \{n | \tilde{\alpha}_n = 0, n = 1, \cdots, M^{\mathrm{RBG}}\}$, i.e., the set of null elements' indexes in $\tilde{\boldsymbol{\alpha}}$. The first step is to set $\beta_n^* = 0$ for all $n \in \mathcal{N}^0$. This operation ensures optimality

[4] We follow the convention that $0 \log \frac{0}{0} = 0$, $0 \log \frac{0}{c} = 0$ and $c \log \frac{c}{0} = +\infty$, where c is a positive constant.

Algorithm 3 Optimal Solution to OPT-β

Initialize $\mathcal{N} := \emptyset$, $\tilde{\mathcal{N}} := \mathcal{N}^+$, $\psi := 1$;
while $\tilde{\mathcal{N}} \neq \emptyset$ **do**
  Determine $\mathcal{N}' := \left\{ n \in \tilde{\mathcal{N}} \middle| \tilde{\alpha}_n/\psi > \gamma_n \frac{M^{\mathrm{SC}}}{L_{\mathrm{URLLC}}} \right\}$;
  if $\mathcal{N}' = \emptyset$ **then**
    for each n in $\tilde{\mathcal{N}}$ **do**
      $\beta_n^* := \tilde{\alpha}_n/\psi$;
    end for
    $\tilde{\mathcal{N}} := \emptyset$;
  else
    for each n in $\mathcal{N}'$ **do**
      $\beta_n^* := \gamma_n \frac{M^{\mathrm{SC}}}{L_{\mathrm{URLLC}}}$;
    end for
    $\tilde{\mathcal{N}} := \tilde{\mathcal{N}} \backslash \mathcal{N}'$;
    $\mathcal{N} := \mathcal{N} \cup \mathcal{N}'$;
    $\psi := \sum_{n \in \tilde{\mathcal{N}}} \tilde{\alpha}_n \Big/ \left(1 - \sum_{n \in \mathcal{N}} \gamma_n \frac{M^{\mathrm{SC}}}{L_{\mathrm{URLLC}}}\right)$
  end if
end while
return $\boldsymbol{\beta}^*$;

w.r.t KL divergence according to (4.3).

(ii) Denote $\mathcal{N}^+ = \{n | \tilde{\alpha}_n > 0, n = 1, 2, \cdots, M^{\mathrm{RBG}}\}$, i.e., the set of indexes of nonzero entries in $\tilde{\boldsymbol{\alpha}}$. For all $n \in \mathcal{N}^+$, we determine the optimal β_n^*'s by solving the following optimization problem:

$$
\begin{aligned}
&\textbf{OPT-}\boldsymbol{\beta} \\
&\text{minimize} \quad D^{\mathrm{KL}}(\tilde{\boldsymbol{\alpha}} \parallel \boldsymbol{\beta}) \\
&\text{subject to} \quad \beta_n \leq \gamma_n \frac{M^{\mathrm{SC}}}{L_{\mathrm{URLLC}}}\,, \quad n \in \mathcal{N}^+ \\
&\qquad\qquad \beta_n \geq 0\,, \qquad\qquad n \in \mathcal{N}^+ \\
&\qquad\qquad \sum_{n \in \mathcal{N}^+} \beta_n = 1\,.
\end{aligned}
$$

In OPT-β, $\tilde{\alpha}_n$'s and γ_n's are constant inputs. Due to the masking and small positive

disturbance applied to $\tilde{\boldsymbol{\alpha}}$ by the post-processor g, we have $\tilde{\alpha}_n > 0$ and $\gamma_n > 0$ for all $n \in \mathcal{N}^+$, and $\sum_{n\in\mathcal{N}^+} \gamma_n = 1$. Then the optimal solution to OPT-β can be obtained by using Algorithm 3. The optimality of Algorithm 3 can be proved by checking KKT conditions since OPT-β is a convex optimization problem. Algorithm 3 has a computational complexity of $O(M^{\text{RBG}})$ and requires no more than M^{RBG} iterations to determine $\boldsymbol{\beta}^*$. Although OPT-β can be solved using standard convex optimizers, our Algorithm 3 uses far fewer number of iterations and thus is more suitable for real-time execution.

Optimal solution $\boldsymbol{\beta}^*$ will be used for actual URLLC preemption in the current TTI.

4.7 Design of the Learning Plane

The goal of learning plane is to optimize approximate algorithm μ using a DRL-based approach. This section presents our design of the learning plane.

4.7.1 Challenge and Proposed Approach

Currently, the most widely used DRL method to address resource allocation problems in wireless networks is the so-called *deep Q-learning* (DQL) [100, 101, 102]. DQL was developed for problems with *discrete* action space with a small number of possible solutions. In each learning iteration, DQL attempts to find an action that has the maximum expected return among all possible actions. That is, DQL requires solving a maximization problem over a discrete action domain in each iteration.

Unfortunately, DQL cannot be used to solve our URLLC preemption problem. The main reason is that our space of all possible preemption solutions is prohibitively large.

L_{URLLC} is typically on the order of hundreds (see footnote 3) and there is a prohibitively large number of different ways to preempt SCs from eMBB users in each TTI. For example, consider that there are 10 eMBB users scheduled in a TTI. For a URLLC packet requiring 300 SCs, the total number of preemption solutions is $\binom{300+10-1}{10-1} \approx 6.3 \times 10^{16}$ (including infeasible solutions).[5] It is simply impractical to apply DQL to such an enormous discrete action space as the computational time to solve all the maximization problems (one for each learning iteration) is prohibitively large.

To mitigate this problem, we employ a different approach. Instead of using (exact) integer numbers for SC preemption from eMBB users, as described in Section 4.6, we use a vector $\boldsymbol{\beta}$ whose entries are fractions within $[0, 1]$ to indicate what *proportion* of SCs for a URLLC transmission should be preempted from each eMBB user. Since L_{URLLC} is large, the space of all possible $\boldsymbol{\beta}$'s is "dense". For instance, for $L_{\text{URLLC}} = 300$, the ℓ^2-norm gap between two adjacent $\boldsymbol{\beta}$'s (with difference of one SC allocation) is $\sqrt{2(\frac{1}{300})^2} = 4.7 \times 10^{-3}$. With this level of closeness, we can consider the space for all $\boldsymbol{\beta}$'s as approximately continuous. For any solution within this approximate continuous space, we can easily find a feasible integral preemption action by adjusting a small number of SCs. This prompts us to consider the DDPG method for learning algorithm μ, which turns out to be an excellent choice for this problem. This is because unlike DQL, DDPG was designed to address problems with large and continuous action domains [85, 104]. In each learning iteration, instead of solving a maximization problem as in DQL, DDPG uses deep function approximator μ to find preemption action. In the next section, we offer essential background on DDPG, laying the foundation for our own contributions in Section 4.7.4.

[5] Finding a solution for preempting L_{URLLC} SCs from N eMBB users is equivalent to determining a non-negative integer solution to the equation: $\sum_i^N x_i = L_{\text{URLLC}}$. It has been shown that the total number of such solutions is $\binom{L_{\text{URLLC}}+N-1}{N-1}$ (Proposition 6.2 in [103]).

4.7.2 DDPG in a Nutshell

DDPG is a model-free DRL method that was developed to solve problems with high-dimensional continuous action domains. DDPG is based on a key notion of "actor" and "critic." An "actor" is a deep function approximator that takes an observed input instance and outputs an action. A "critic" is another deep function approximator that predicts the expected return of an action under a given input instance. The role of the critic is to offer an *approximate* objective function of the underlying optimization problem in each learning iteration. The ultimate objective of DDPG is to obtain an optimal actor algorithm (neural network) that is able to maximize the expected sum of discounted rewards starting from the first iteration, i.e., $E\left[\sum_{t=1}^{T} d^{t-1} r(t)\right]$, where $d \in [0, 1]$ is a discount factor and T represents the total number of iterations.

More formally, denote Q as the critic function approximator. Q is used for predicting the following expected return in each iteration:

$$Q(\boldsymbol{s}(t), \boldsymbol{\alpha}(t)) = E\Big[\sum_{j=t}^{T} d^{j-t} r(j) \Big| \boldsymbol{s}(t), \boldsymbol{\alpha}(t)\Big]. \tag{4.4}$$

Under DDPG, (4.4) is called the action-value. It represents the expected return we can obtain by taking an action $\boldsymbol{\alpha}(t)$ with an input $\boldsymbol{s}(t)$, under the assumption that the current actor algorithm will be used for all future iterations.

The learning process of DDPG is to optimize parameters in actor and critic neural networks through a large number of iterations. Denote Φ as the set of trainable parameters in Q (weights and biases similar to those in Θ). In each iteration, the actor takes an action for a given input instance. Then a reward signal is received for this action. This action execution instance is then stored in a replay buffer. Next, a subset of K stored execution instances are randomly sampled from the replay buffer. Using these samples, the critic network Q

is updated based on received reward signals to improve its prediction accuracy. Then the updated Q is used to optimize the actor. Parameters within the actor network are updated by taking a small step along the gradient of Q w.r.t. these parameters.

To ensure the learning can converge eventually (i.e., obtain a stable and maximized return), DDPG employs two additional neural networks, called *target networks* to help stabilize the learning of Q. Specifically, based on the Bellman equation [85], action-value (4.4) can be rewritten as:

$$Q(\boldsymbol{s}(t), \boldsymbol{\alpha}(t)) = E\Big[d \cdot Q(\boldsymbol{s}(t+1) + r(t), \boldsymbol{\alpha}(t+1))\Big|\boldsymbol{s}(t), \boldsymbol{\alpha}(t)\Big]. \tag{4.5}$$

The target networks are used to approximate (substitute) $Q(\boldsymbol{s}(t+1), \boldsymbol{\alpha}(t+1))$ in (4.5). Then the sum of reward $r(t)$ and the action-value for iteration $t+1$ that is predicted by target networks provides a "target" for the update of critic network Q. The use of target networks can effectively stabilize the learning of Q. However, the downside is that the learning process will slow down, as target networks are constrained to have very small update in each iteration (soft update with a coefficient $\tau \ll 1$ [85]).

4.7.3 Convergence Problem

For our URLLC preemption problem, we find that if we follow DDPG as it is, then the learning is extremely slow to converge. In our experiments, we found that there was no significant improvement in objective (i.e., reducing the loss of eMBB utility) after more than half a million iterations. The reason behind this is that the learning for critic Q is too slow. The accuracy of Q for predicting action-value (4.4) does not show much improvement as the number of learning iterations increases. Without an accurate critic Q, DDPG cannot converge in learning algorithm μ. Further, the inclusion of target networks in DDPG also

slows down the learning progress since target networks are constrained to be updated very slowly using a coefficient $\tau \ll 1$. In the next section, we present our design to mitigate this convergence problem.

4.7.4 Our Design

Our design of the learning plane is shown in Fig. 4.2. It retains an actor-critic structure similar to DDPG (with the actor being neural network μ described in Section 4.6.2 and the critic being another neural network Q) but with some significant differences:

(i) Based on our observation of some intrinsic properties associated with the URLLC preemption problem, we propose to simplify the learning objective of DDPG and the action-value that critic Q needs to predict. We show that these simplifications can achieve a much faster and more stable convergence compared to original DDPG.

(ii) Following the above simplifications, we propose to remove the target networks in DDPG from our design. This will eliminate the delay associated with target networks but will not hamper the stability of learning process.

The rest of this section is organized as follows. We first describe our design of reward function, which serves as an input to the learning plane. Then we describe how to design a learning method that can speed up convergence. A summary of the proposed learning method is given in Algorithm 4.

Construction of Reward Function The reward function in each TTI is a component of the action-value (4.4) that critic Q needs to approximate. There are two aspects that must be considered when constructing a reward function for our problem:

(i) First and foremost, the reward function must offer a critical assessment (evaluation) for a preemption instruction $\boldsymbol{\alpha}(t)$ in terms of its impact on eMBB utility;

(ii) Second, the reward function should include a penalty for any infeasibility incurred in μ's output $\boldsymbol{\alpha}(t)$.

For the first consideration (critical assessment), we need to determine the loss of eMBB utility in a TTI due to URLLC preemption. Denote $U_n(t)$ as eMBB user n's utility when it can successfully decode the received transmission in TTI t. Denote $U(t)$ as the maximum sum utility of all eMBB users scheduled for TTI t. Then we have $U(t) = \sum_n U_n(t)$. If all eMBB transmissions are successful, then the maximum $U(t)$ is achieved. But in the presence of URLLC preemption, some eMBB user(s) may not be able to decode its received signal, resulting in a loss of eMBB utility.

Denote $\boldsymbol{\epsilon}(t)$ as a vector indicating whether or not each eMBB transmission is successfully decoded. For each eMBB user n, we have

$$\epsilon_n(t) = \begin{cases} 0, & \text{if transmission to eMBB user } n \text{ is successfully decoded;} \\ -1, & \text{otherwise.} \end{cases}$$

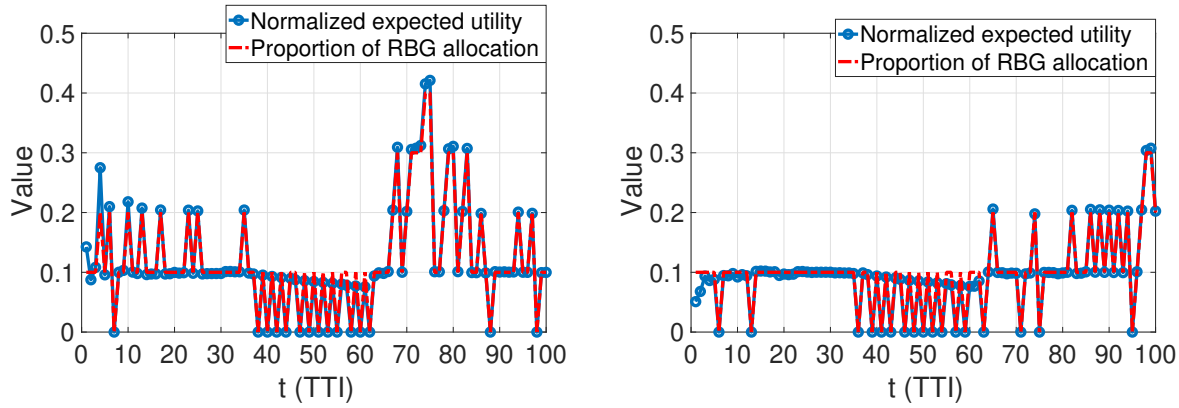
Then in TTI t, the normalized eMBB utility loss is: $\sum_n \epsilon_n(t) U_n(t)/U(t)$.

Denote $r^{\mathrm{LU}}(t)$ as the reward for the eMBB utility achieved in TTI t. $r^{\mathrm{LU}}(t)$ is defined as

$$r^{\mathrm{LU}}(t) = \sum_n \frac{U_n(t)}{U(t)} \cdot \epsilon_n(t) \ .$$

Then the range of $r^{\mathrm{LU}}(t)$ is between $[-1, 0]$. For examples, if all eMBB transmissions are successful, then $r^{\mathrm{LU}}(t) = 0$; if all eMBB transmissions fail, then $r^{\mathrm{LU}}(t) = -1$.

If we were to use this reward function, then we would have to add additional information



(a) eMBB user 1. (b) eMBB user 2.

Figure 4.4: Relationship between the proportion of RBGs allocated to an eMBB user and its normalized expected PF utility.

$(U_n(t)/U(t)$'s) to the input of μ. This is undesirable as it requires to add more intermediate layers in μ, which will result in a longer execution time. Instead, we find that we can obtain a good approximation for $r^{\mathrm{LU}}(t)$ by using information already included in the input $\boldsymbol{s}(t)$. Through experiments, we found that the normalized expected utility $U_n(t)/U(t)$ of an eMBB user is highly correlated with the proportion of RBGs allocated to it, i.e., $\gamma_n(t)$. The reason is that a user with a higher expected utility is more likely to be allocated with more RBGs. This correlation has been verified by our experimental results shown in Fig. 4.4, where we run a PHY-MAC NR simulator (see Section 4.8.1) with proportional-fairness (PF) scheduling for 10 eMBB users and show the results for the first and second users for 100 TTIs. In Fig. 4.4, we can see that $\gamma_n(t)$ has a nearly perfect match to $U_n(t)/U(t)$. Based on this insight, we propose to use

$$r^{\mathrm{LU}}(t) = -\sum_n \epsilon_n(t)\gamma_n(t) \tag{4.6}$$

as a reward function. This innovative design allows us to avoid bringing any additional input to μ.

For the second consideration (i.e., to penalize infeasibilites in $\boldsymbol{\alpha}(t)$), we propose to include

penalties for both the redundancies in $\boldsymbol{\alpha}(t)$ and the KL divergence $D^{\mathrm{KL}}(\tilde{\boldsymbol{\alpha}}(t) \parallel \boldsymbol{\beta}^*(t))$. For the redundancies in $\boldsymbol{\alpha}(t)$, we introduce a penalty

$$r^{\mathrm{RE}}(t) = -\sum_{n \in \mathcal{N}^0(t)} \alpha_n(t). \tag{4.7}$$

For KL divergence $D^{\mathrm{KL}}(\tilde{\boldsymbol{\alpha}}(t) \parallel \boldsymbol{\beta}^*(t))$, we use the penalty

$$r^{\mathrm{KL}}(t) = -D^{\mathrm{KL}}(\tilde{\boldsymbol{\alpha}}(t) \parallel \boldsymbol{\beta}^*(t)). \tag{4.8}$$

The goal of (4.7) and (4.8) is to reduce the distance between algorithm μ's output $\boldsymbol{\alpha}(t)$ and the final feasible preemption solution $\boldsymbol{\beta}^*(t)$ through the learning process. This design helps to minimize the loss of information caused by g and f.

Putting (4.6), (4.7) and (4.8) together, we construct the complete reward function $r(t)$ as follows:

$$r(t) = w^{\mathrm{UL}} r^{\mathrm{LU}}(t) + w^{\mathrm{RE}} r^{\mathrm{RE}}(t) + w^{\mathrm{KL}} r^{\mathrm{KL}}(t), \tag{4.9}$$

where w^{UL}, w^{RE}, and w^{KL} are positive weights.

Mechanisms to Accelerate Convergence To address the convergence problem, we take a closer look at our URLLC preemption problem. The objective of our problem is to minimize the loss of eMBB utility due to URLLC preemption. To characterize eMBB utility, let's consider some of the most-widely used schedulers for eMBB, such as proportional-fairness (PF) [49, 50, 52, 55, 70], maximum throughput (MT), and other weighted-rate based schedulers [70]. A common characteristic of these eMBB schedulers is that they are mostly greedy algorithms that aim to maximize the eMBB utility in each TTI independently. An attractive feature of these greedy algorithms is that they have low computational complexities and thus are suitable for real-time execution. In this chapter, we consider the PF scheduler

[50, 52, 55, 70] as an example. The same design of learning method also applies to other greedy eMBB schedulers.

The PF scheduler is known for its ability to strike a tradeoff between fairness and total cell throughput among the users. Under PF scheduling, the utility function is defined as $\sum_n \log \bar{C}_n$, where $\bar{C}_n$ denotes user n's long-term average data rate. It has been proved that an optimal PF scheduling solution is to allocate RBGs with the following objective in each TTI t:

$$\text{maximize} \quad U(t) = \sum_n \frac{\hat{C}_n(t)}{\bar{C}_n(t-1)}, \tag{4.10}$$

where $\hat{C}_n(t)$ is user n's expected data rate, and $\bar{C}_n(t-1)$ is user n's exponentially smoothed average data rate. $\bar{C}_n(t-1)$ is updated as:

$$\bar{C}_n(t-1) = \eta \cdot C_n(t-1) + (1-\eta) \cdot \bar{C}_n(t-2), \tag{4.11}$$

where $C_n(t-1)$ is user n's achieved data rate in TTI $(t-1)$, and η is a small positive coefficient (e.g., 0.01). It has been shown that this per-TTI scheduling maximizes PF utility when $t \to \infty$ [49].

For our URLLC preemption problem, to minimize the loss of eMBB utility, it is sufficient to minimize the loss of utility $U(t)$ in each TTI. Following the same proof for per-TTI scheduling (4.10), it is easy to show that this approach is asymptotically-optimal for minimizing the loss of eMBB utility. Note that this property holds for MT scheduler and other weighted-rate based schedulers as well. For these schedulers, maximizing per-TTI utility metrics is globally optimal.

Now the question to ask is: how can we exploit this property to design a learning method? Our first observation is that the learning objective can be reduced to: maximize $E[r(t)]$,

Algorithm 4 A Method for Learning μ

Initialize Θ and Φ randomly for actor network μ and critic network Q;
Reset replay buffer;
Receive an eMBB state vector $\boldsymbol{s}(1)$;
for each TTI $t = 1, 2, \cdots, T$ **do**
    Obtain the output of μ: $\boldsymbol{\alpha}(t) \leftarrow \mu(\boldsymbol{s}(t)|\Theta)$;
    Apply post-processor g: $\tilde{\boldsymbol{\alpha}}(t) \leftarrow g(\boldsymbol{\alpha}(t))$;
    Apply translator f: $\boldsymbol{\beta}(t) \leftarrow f(\tilde{\boldsymbol{\alpha}}(t), \boldsymbol{s}(t))$;
    Use $\boldsymbol{\beta}(t)$ for URLLC puncturing in TTI t;
    Receive $r(t)$ and new eMBB state vector $\boldsymbol{s}(t+1)$;
    Store $(\boldsymbol{s}(t), \boldsymbol{\alpha}(t), r(t))$ into replay buffer;
    Randomly sample a mini-batch of K results $(\boldsymbol{s}(i), \boldsymbol{\alpha}(i), r(i))$ from replay buffer;
    Update Q through (4.13);
    Update μ using the sample gradient (4.14);
end for
return Learned μ;

That is, it is sufficient to maximize the expected reward in each TTI without accounting for future rewards. As discussed earlier, a key component in $r(t)$ is the reward for the achieved eMBB utility $U(t)$. Thus, to minimize the eMBB utility loss, we only need to maximize $E[r(t)]$ for each TTI. As a result, the action-value (4.4) is simplified to:

$$Q(\boldsymbol{s}(t), \boldsymbol{\alpha}(t)) = E\Big[r(t)\Big|\boldsymbol{s}(t), \boldsymbol{\alpha}(t)\Big] . \tag{4.12}$$

These simplifications accelerate convergence on the learning plane in two aspects. First, the learning for critic Q becomes much easier and faster because Q is no longer required to predict expected return in future TTIs. Second, there is no need to include target networks since action-value (4.12) only involves expected reward in the current TTI. Moreover, the removal of target networks will not impact stability of learning.

Learning Method Algorithm 4 summarizes our proposed learning method based on the action-value (4.12). For initialization, parameters in neural networks μ and Q are randomly generated. Then the memory is allocated for the replay buffer. At the beginning of learning, an initial eMBB instance $\boldsymbol{s}(1)$ for TTI $t = 1$ is received. Then the learning process of

Algorithm 4 goes through a *for* loop with a total number of T iterations (Line 4-13). In each iteration, an algorithm execution instance $(\boldsymbol{s}(t), \boldsymbol{\alpha}(t), r(t))$ is obtained from scheduling plane and stored in the replay buffer (Line 5-9). Then a subset of K samples are randomly selected from the replay buffer. These samples are used for updating parameters in critic network Q and algorithm μ. Q is updated by solving:

$$\text{minimize} \quad \frac{1}{K}\sum_{i=1}^{K}\left(Q(\boldsymbol{s}(i), \boldsymbol{\alpha}(i)|\Phi) - r(i)\right)^2. \tag{4.13}$$

An efficient implementation of (4.13) is to update parameters in Q (Φ) by taking a small step along the gradient of the minimization objective w.r.t. Q's parameters. Note that in (4.13), the update of Q is only based on rewards $r(i)$'s without any target network. This corresponds to our simplification for action-value in (4.12). After updating Q, parameters in algorithm μ (Θ) are updated using the following sample gradient (SG):

$$SG = \frac{1}{K}\sum_{i=1}^{K} \nabla_{\Theta}\mu(\boldsymbol{s}|\Theta)|_{\boldsymbol{s}(i)} \cdot \nabla_{\boldsymbol{\alpha}} Q(\boldsymbol{s}, \boldsymbol{\alpha}|\Phi)|_{\boldsymbol{s}=\boldsymbol{s}(i), \boldsymbol{\alpha}=\mu(\boldsymbol{s}(i))}; \tag{4.14}$$

When the *for* loop completes, a learned algorithm μ is returned.

Stability of Learning From (4.13), we can see that the learning of action-value (4.12) is very similar to supervised learning. Theoretically, neural network Q is used to approximate the expected reward $E\Big[r(t)\Big|\boldsymbol{s}(t), \boldsymbol{\alpha}(t)\Big]$ in each TTI t. In practical implementation, we use the reward $r(t)$ as an unbiased estimation for $E\Big[r(t)\Big|\boldsymbol{s}(t), \boldsymbol{\alpha}(t)\Big]$. During the learning process, network Q is trained with tuples ($\boldsymbol{s}(i)$, $\boldsymbol{\alpha}(i)$, $r(i)$) for a large number of TTIs (i.e., (4.13)). The goal is to have a network Q that can precisely predict expected reward $E\Big[r(t)\Big]$ given an input instance ($\boldsymbol{s}(t)$, $\boldsymbol{\alpha}(t)$). Such learning of Q resembles a supervised learning process (with a large amount of training data ($\boldsymbol{s}(i)$, $\boldsymbol{\alpha}(i)$, $r(i)$)). Moreover, since the learning target for network Q is simply $r(t)$ and Q is not used to calculate the target value, our design

Table 4.2: MCS levels and measured working SNRs (dB) under BLER = 0.1.

MCS level	1	2	3	4	5	6	7	8	9	10
Modulation	QPSK				16QAM				64QAM	
Code rate	0.10	0.21	0.33	0.42	0.29	0.39	0.49	0.58	0.46	0.55
Working SNR (dB)	-1.4	1.3	4.2	6.1	9.6	11.8	14.0	16.2	18.3	21.0

eliminates the divergence/stability issue in DDPG.

4.8 Implementation

This section presents our implementation of DEMUX in a simulated 5G NR network environment. Our implementation has the following two components:

(i) A PHY-MAC NR simulator supporting dynamic eMBB/URLLC multiplexing with URLLC preemption;

(ii) A DRL module implementing our learning method in Section 4.7.4.

We employ object-oriented programming to implement both components. Each component involves a number of classes. Procedures within the closed-loop across scheduling and learning planes (see Fig. 4.2) are implemented as different methods encapsulated in these classes. In the rest of this section, we give details of our implementation.

4.8.1 PHY-MAC NR Simulator

Our NR simulator implements NR PHY and MAC layer functions that are most relevant to the URLLC preemption problem. This simulator is built using MathWorks' 5G Toolbox

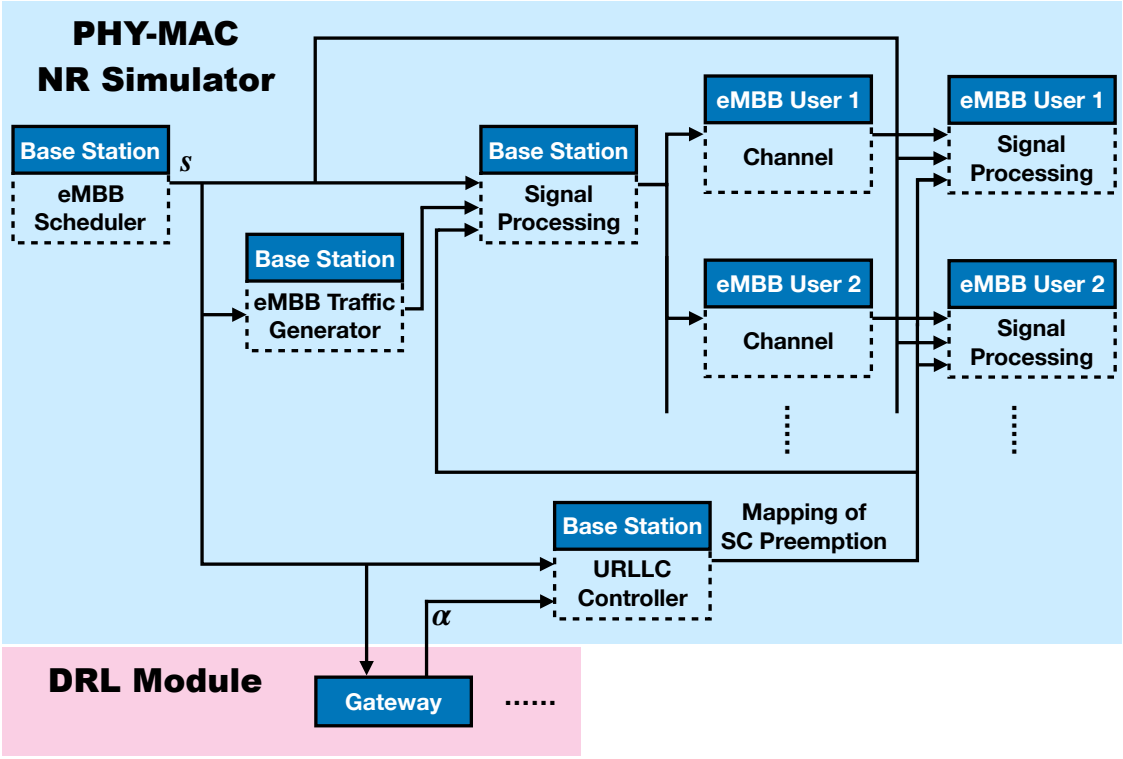


Figure 4.5: An implementation diagram of our NR simulator.

[105], which offers standard-compliant signal processing functions for 5G NR end-to-end communications, including channel coding, modulation, OFDM modulation, and multi-path channel fading. In addition to these signal processing functions, we implement modules to support SC preemption for URLLC packet transmissions, simultaneous multi-link eMBB transmissions, RBG scheduling and MCS selection for eMBB users, and TCP/IP socket based communication with the DRL module.

Fig. 4.5 illustrates our implementation of the NR simulator. As shown in this figure, the NR simulator has a *Base Station* (BS) class and an *eMBB user* class. The BS class contains methods for per-TTI downlink processing procedures including eMBB scheduling, URLLC control, data generation, and transmitter-side signal processing. The eMBB user class contains methods for channel fading and noise generation, and receiver-side signal processing. A simulated NR cell instance consists of an object of the BS class and multiple

objects of the eMBB user class. This characterizes multiple communication links between the BS and eMBB users.

The BS class has the following methods:

- **eMBB Scheduler** This method schedules eMBB transmissions in each TTI and assigns RBGs and MCS level for each transmission. For RBG allocation, we employ the PF scheduler described in Section 4.7.4. In each TTI, RBGs are allocated iteratively to eMBB users based on a *copy* of long-term average rates $\bar{C}_n(t)$'s that is updated after allocating each RBG as in (4.11). The original $\bar{C}_n(t)$'s are updated after all RBGs are allocated in each TTI. For eMBB MCS selection, we consider 10 MCS levels given in Table 4.2. Each MCS level requires a minimum working SNR to ensure that the block error rate (BLER) is no greater than a target threshold. The working SNRs for 0.1 BLER (typical value for eMBB) of these MCS levels are measured through experiments using our NR simulator. The measurement is done for CDL-C multi-path channel fading with 300 ns delay spread on 25 RBs. For each eMBB user, the BS selects an MCS level so that the user's average SNR is no less than the MCS's working SNR. This method returns an eMBB instance vector $\boldsymbol{s}(t)$. In particular, $\boldsymbol{s}(t)$ will be sent to the DRL module via Gateway. The Learning Agent will use $\boldsymbol{s}(t)$ to produce a preemption instruction $\boldsymbol{\alpha}(t)$ based on algorithm μ.

- **URLLC Controller** This method controls URLLC transmissions and preemptive puncturing in each TTI. Input to this method includes eMBB instance $\boldsymbol{s}(t)$ and puncturing instruction $\boldsymbol{\alpha}(t)$ from DRL module. URLLC controller performs the following tasks. First, it obtains a feasible preemption solution $\boldsymbol{\beta}(t) = (f \circ g)(\boldsymbol{\alpha}(t))$ using g and f as described in Section 4.6.3. Second, URLLC packets are generated using a Poisson process with arrival rate λ (arrivals/TTI). λ is a configurable parameter of

the simulator. Arrived URLLC packets are stored in a first-in-first-out buffer. Third, each URLLC packet is scheduled to transmit in the earliest available mini-slot after its arrival. In each mini-slot, at most one URLLC packet is transmitted. Fourth, the preemption of SCs for each URLLC transmission is determined based on solution $\boldsymbol{\beta}(t)$. As discussed in Section 4.2, the preemption on each eMBB transmission proceeds from the SC with the lowest index that is allocated to it. Finally, a frequency-time resource mapping for preemption is generated, which indicates OFDM symbols and SCs that are punctured by URLLC in the current TTI. The preemption mapping is returned as the output of this method.

- **eMBB Traffic Generator** This method generates data bits for scheduled eMBB transmissions. For an eMBB transmission, the number of data bits within its transport block (TB) depends on the number of allocated RBGs and assigned MCS level. In general, a TB contains more data bits with more RBGs and a higher MCS level. Data bits within each TB are first generated as a random binary sequence. Then cyclic redundancy check (CRC) bits are computed and attached to TBs. These CRC bits will be used for checking whether each eMBB transmission succeeds.

- **Transmitter-Side Signal Processing** This method implements downlink signal processing chain at the BS. Input to this method includes eMBB instance $\boldsymbol{s}(t)$, preemption mapping from URLLC Controller method, and TBs generated by eMBB Traffic Generator. The processing procedures include LDPC encoding, rate matching, scrambling, modulation (QPSK, 16QAM or 64QAM), resource preemption and OFDM modulation. After modulation, eMBB symbols on preempted resources are flushed out according to the preemption mapping. In each TTI, this method processes TBs for all scheduled eMBB transmissions.

The eMBB user class has the following methods:

- **Channel** This method generates frequency and time domain channel fading and noise for an eMBB transmission. It is called by an eMBB user object before user-side signal processing. Input to this method is an eMBB transmission after BS-side signal processing. Practical multi-path channel fading models such as the Cluster Delay Line (CDL) model and the Tap Delay Line (TDL) model [40] are employed in this method.
- **Receiver-Side Signal Processing** This method implements signal processing chain at eMBB user side. Input to this method includes $\boldsymbol{s}(t)$, preemption mapping, and returned eMBB transmission from the Channel method. The processing procedures include OFDM demodulation, soft demodulation, preemption flushing, descrambling, rate recovery, and LDPC decoding. The operation of preemption flushing is to nullify corrupted bits after soft demodulation according to the preemption mapping. This operation prevents error propagation during LDPC decoding. After decoding, the CRC check results indicate whether each eMBB transmission is successful.

4.8.2 DRL Module

We implement the DRL module using TensorFlow version 1.13.1 [106]. The DRL module involves a *Gateway* class and a *Learning Agent* class. The Gateway class offers a communication interface between NR simulator and DRL module using TCP/IP socket. During our experiment, the transfer of eMBB instance $\boldsymbol{s}(t)$ and reward $r(t)$ from NR simulator to DRL module and the transfer of preemption instruction $\boldsymbol{\alpha}(t)$ from DRL module to NR simulator all go through an object of this class.

The Learning Agent class implements our learning method in Section 4.7.4. Both neural networks μ and Q are contained in this class. Tasks for this class include: (i) learning μ using

Algorithm 4, and (ii) executing learned μ with input instance $\boldsymbol{s}(t)$ received via Gateway. It has the following methods:

- **Model Setup** This method sets up neural networks μ and Q, optimizers used for updating parameters in μ and Q, and replay buffer for storing algorithm execution instances. Input to this method is a configuration profile specifying the numbers of layers and neurons per layer, nonlinearity for each neuron, input and output formats for networks μ and Q, and other configuration settings. The optimizers for Q and μ first compute gradients of Q w.r.t. parameters in Θ and Φ, respectively, and then update parameters in Q and μ along the gradients.

- **Learner** This method implements Algorithm 4 for learning algorithm μ. We employ a very flexible implementation of the learning method: In each learning iteration, algorithm μ is executed for a number of consecutive TTIs. Then parameters in Q and μ are updated for multiple times using different subsets of samples. Settings in this implementation (i.e., how many algorithm executions and parameter updates per iteration) can be tuned through experiment to achieve an optimized learning performance.

- **Actor** This method takes an eMBB instance $\boldsymbol{s}(t)$ as input (obtained via Gateway) and uses learned algorithm μ to produce a preemption instruction $\boldsymbol{\alpha}(t)$, which is then sent to NR simulator through Gateway.

4.8.3 Putting Our Implementation to Work

We now describe how to use our NR simulator and DRL module for learning algorithm μ. The first step is to set up an experiment instance consisting of an object of BS class, a number of objects of eMBB user class, an object of Gateway class, and an object of Learning Agent class. Since in each TTI at most M^{RBG} eMBB users can be scheduled (see Section 4.6.2), we

only need to generate M^{RBG} eMBB user objects in the experiment instance. To fully explore the space of all possible eMBB instances $\boldsymbol{s}(t)$'s, we randomly allocate RBGs on the channel to eMBB users in each TTI. MCS level for each eMBB user is also randomly selected from Table 4.2 with average received SNRs equal to working SNRs of the selected MCS levels. Thus every possible eMBB instance $\boldsymbol{s}(t)$ has the same probability to appear during learning, leading to a full exploration of input space for μ.

Then we run the learning process shown in Fig. 4.2 for T iterations (e.g., on the order of 10^5 or higher). During learning, we turn off multi-path channel fading and noise effects in the NR simulator (in eMBB user objects). The reason is that with fading and noise, an eMBB transmission could fail even without URLLC preemption, making it hard to deduce whether or not a transmission failure is due to the underlying preemption solution. Our experiment results in Section 4.9 show that algorithms learned under this approach perform well in all network scenarios when different channel fading models are present.

Another issue for learning is that with Poisson packet arrivals, the number of URLLC transmissions in each TTI is a random number between zero and the number of mini-slots per time slot. Thus eMBB transmission failure not only depends on the preemption solution $\boldsymbol{\beta}(t)$ but also on the number of URLLC transmissions in a TTI. This again makes it difficult to learn the quality of a preemption solution. We propose the following approach to address this issue. In realistic cells, the URLLC packet arrival rate λ always has a finite range. Assume that there are N mini-slots per TTI. The BS should perform proper URLLC traffic control to prevent overflow of URLLC packets, e.g., ensure that no more than N packets will arrive in each TTI. Then the range of λ is $[0, N]$. Based on this knowledge, one can uniformly sample a set of discrete arrival rates from $[0, N]$ (e.g., $\{1, 2, \cdots, N\}$) and learn a separate actor network μ offline under each sampled arrival rate. In the learning phase, we fix the number of URLLC transmissions in each TTI to the given arrival rate. Since each

network μ is trained under a given *known* arrival rate, it does not need to learn the packet arriving pattern. The sole objective of learning is let μ approximate the optimal puncturing solution under the given arrival rate.

During cell operating time, the first task for the BS is to estimate the URLLC packet arrival rate empirically from data collected in the cell. This estimation can be done through statistical methods and is independent of the learning of actor networks μ's. Then the BS scheduler can choose a network μ (learned under a known arrival rate) that is closest to the cell's actual URLLC traffic condition. The chosen network will perform well if it matches to the estimated arrival rate.

Therefore, we do not need to explicitly include the arrival rate into the input layer of network μ since a separate network is learned under a given arrival rate. Moreover, there is no need to include the arrival rate in the reward function.

4.9 Validation

This section validates the performance of DEMUX based on our implementation in Section 4.8. All experiments are done on a Dell desktop computer with an Intel Xeon E5-2687W v4 CPU (3.0 GHz).

4.9.1 Experimental Settings

NR Simulator We use our NR simulator to set up a downlink NR cell environment with a BS, a number of eMBB users and a given URLLC traffic load. Key parameter settings of the cell environment are given in Table 4.3.

All eMBB users are uniformly and randomly distributed within the cell's coverage. The

Table 4.3: Key parameter settings in the the NR simulator.

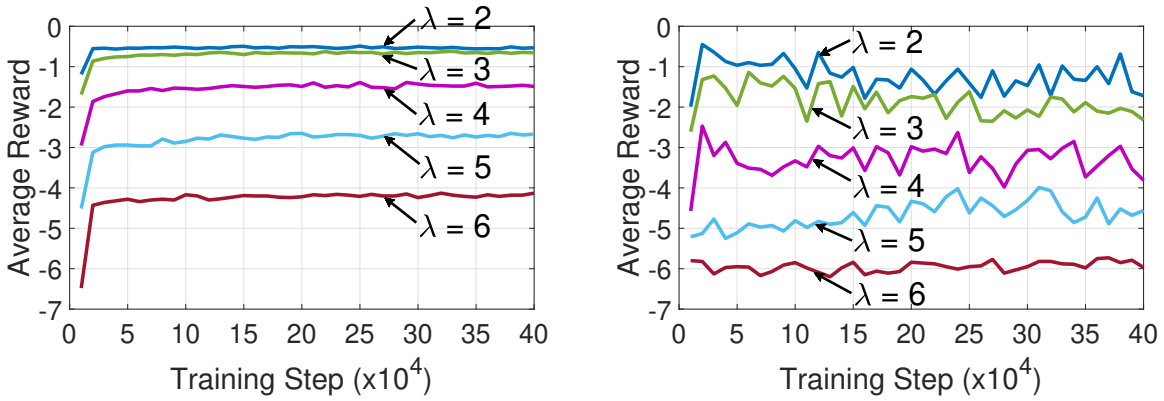
NR numerology	Numerology 1 (30 kHz SC spacing)
System bandwidth	20 MHz (50 RBs in total, 5 RBs/RBG)
Carrier frequency	4 GHz
Cell radius	1000 m
eMBB config.	1 TTI = 1 time slot (14 OFDM symbols) Full buffer traffic model
URLLC config.	1 mini-slot = 2 OFDM symbols Poisson arrival of 50-byte packets
Fading channel	CDL-C/TDL-C with 300 ns RMS delay spread [40]
Pathloss model	3D UMa NLOS [40]
Channel estimation	Ideal channel estimation
Antenna config.	1 Tx antenna and 1 Rx antenna
BS Tx power	46 dBm
Noise floor	-91.9 dBm

Poisson URLLC packet arrival rate is chosen from $\{2, 3, 4, 5, 6\}$. For eMBB PF scheduling, the coefficient η used for updating long-term average rates $\bar{C}_n(t)$'s in (4.11) is set to 0.01. MCS level for each eMBB user is selected by finding the highest level in Table 4.2 that can be supported by the user's average received SNR. The MCS selection vector $\boldsymbol{\delta}(t)$ is set to code rates of eMBB transmissions. This ensures all entries of vector $\boldsymbol{s}(t)$ (μ's input) are within $[0, 1]$. We employ QPSK modulation with 1/3 code rate for URLLC transmissions [82]. Under this settings, we have $L_{\text{URLLC}} = 300$. All eMBB and URLLC transmissions are with single transmit and receive antenna and one code word.

We consider two multi-path channel fading models – TDL and CDL, that are widely used for link-level simulations. TDL involves power, delay and Doppler spectrum information for each multi-path channel tap. CDL is a spatial extension of TDL that includes more detailed information such as angle for each cluster of channel taps.

DRL Module For actor neural network μ, we employ a feedforward neural network with 2 fully-connected intermediate layers each with 256 neurons. The rectified non-linearity is used as the activation function for intermediate layers. The input layer is the eMBB instance vector $\boldsymbol{s}(t)$ of size $2M^{\text{RBG}}$. A softmax function is used before the output layer $\boldsymbol{\alpha}(t)$ to ensure that the sum of entries in $\boldsymbol{\alpha}(t)$ equals to one. To ensure a broader exploration of the action space, we add a small random noise to $\boldsymbol{\alpha}(t)$, the output of μ. This action noise is assumed to have Gaussian distribution with zero mean and a standard deviation 0.02. Then all entries in $\boldsymbol{\alpha}(t)$ are clipped to ensure they are within $[0, 1]$ and normalized afterwards.

The critic network Q is also a feedforward neural network with the same intermediate layer structure and activation as μ. The eMBB instance vector $\boldsymbol{s}(t)$ is the input to the first intermediate layer. The output $\boldsymbol{\alpha}(t)$ of μ is additional input to the second intermediate layer. Q's output is a single unconstrained neuron that predicts the action-value. We use the Adam algorithm [107] for learning parameters in Q and μ with a learning rate of 10^{-3}



(a) DEMUX (b) Original DDPG

Figure 4.6: Learning curves of DEMUX and original DDPG under different URLLC packet arrival rates.

per iteration. A subset size $K = 128$ for random sampling and replay buffer size of 10^5 are employed.

4.9.2 Benchmark Comparison

For performance comparison, we implement two other state-of-the-art URLLC preemption schemes on the same NR simulator platform. The first is the RP algorithm proposed in [86]. For each URLLC transmission, RP preempts SCs from each eMBB user in proportion to its RBG allocation, i.e., the preemption solution is $\boldsymbol{\beta}^{\mathrm{RP}}(t) = \boldsymbol{\gamma}(t)$. It was shown in [86] that RP is optimal if the cost function of URLLC preemption is linear.[6]

The second benchmark scheme is FFP that was proposed in 3GPP standards body [87, 88]. This scheme divides the channel bandwidth into multiple non-overlapping and fixed frequency parts. All URLLC packets are transmitted using one specified frequency part. Under our setting $M^{\mathrm{SC}}/L_{\mathrm{URLLC}} = 2$, the channel bandwidth is divided into two parts and all URLLC transmissions will use the first part.

[6] In [86], the authors also proposed an algorithm for convex cost function. But this algorithm requires an explicit closed-form cost function, which is not available in practice.

4.9.3 Results

Convergence of Learning We use our implementation in Section 4.8 to learn an algorithm μ for each URLLC arrival rate $\lambda \in \{2, 3, 4, 5, 6\}$. Weights for the three components in the reward function (4.9) are set to $w^{\mathrm{UL}} = 10$, $w^{\mathrm{RE}} = 3$, and $w^{\mathrm{KL}} = 5$. Other settings are also possible while w^{UL} should generally be greater than w^{RE} and w^{KL} due to the significance of $r^{\mathrm{LU}}(t)$ in the reward function. Fig. 4.6a shows the learning curves of DEMUX over 4×10^5 steps under different arrival rates. Each point on a curve represents the reward value averaged over the past 100 TTIs. We can see that the learning processes of DEMUX converge very quickly and smoothly within 5×10^4.

As a comparison, Fig. 4.6b shows the learning curves of the original DDPG (refer to Section 4.7.2) under the same reward function setting ($w^{\mathrm{UL}} = 10$, $w^{\mathrm{RE}} = 3$, and $w^{\mathrm{KL}} = 5$). In contrast to DEMUX, the learning curves of DDPG do not converge even after 4×10^5 steps and the average rewards fluctuate significantly as the number of steps increases. As we pointed out in Section 4.7.3, the main reason is that DDPG cannot effectively learn an accurate critic Q for our URLLC preemption problem.

Comparison of Performance Objectives In the following experiments, we compare the performance of DEMUX, RP and FFP. We consider two performance objectives:

(i) eMBB users' sum PF utility: $\sum_n \log \bar{C}_n$, which is our optimization objective (i.e., minimizing the loss of eMBB PF utility);

(ii) eMBB cell throughput: $\sum_n \bar{C}_n$, which is an important performance metric that is of major concern to cellular operators.

We show results for 10 and 30 eMBB users, which are typically used for performance evaluation in 3GPP standards body (e.g., Table 6.1.2-1 in [34] for dense urban scenario).

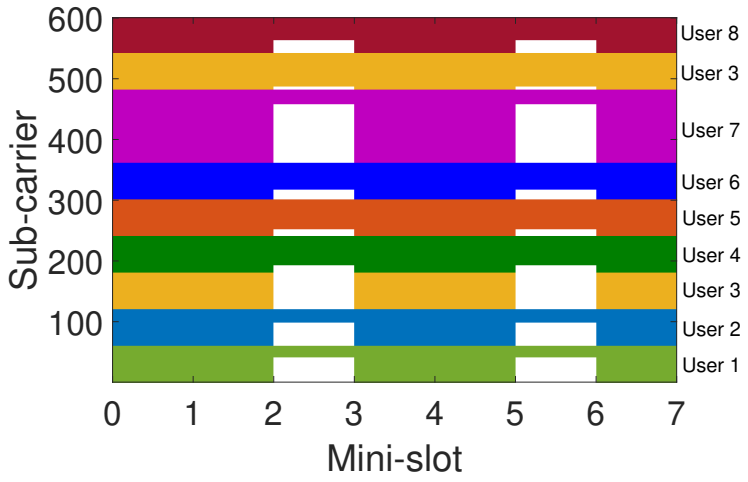
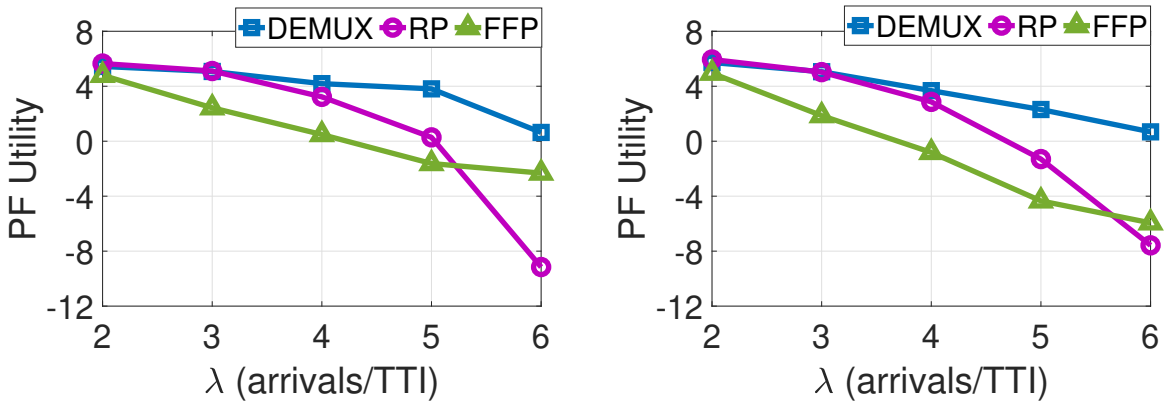


Figure 4.7: An example of URLLC preemption by DEMUX. Resources allocated to each eMBB user are represented by a different color. Resources preempted by URLLC are in white blank.

(i) 10 eMBB Users. We first run experiments with 10 eMBB users under different URLLC traffic loads and channel fading models. Each preemption scheme (DEMUX, RP or FFP) is executed for 10^4 consecutive TTIs under a given URLLC arrival rate $\lambda \in \{2, 3, 4, 5, 6\}$ and fading model.

To understand how URLLC preemption works, in Fig. 4.7, we provide a live example. Following Table 4.3, a mini-slot contains two OFDM symbols and a time slot consists of seven mini-slots. As shown in Fig. 4.7, there are 8 eMBB transmissions scheduled (out of 10 users) in this TTI. The numbers of SCs allocated to the 8 eMBB users are 60, 60, 120, 60, 60, 60, 120, 60, respectively. There are 2 URLLC transmissions occurring at the 3rd and 6th mini-slots, respectively. Under DEMUX, the numbers of SCs preempted from the 8 eMBB users are 43, 37, 66, 9, 8, 15, 96, 26, respectively. In contrast (not shown in Fig. 4.7), FFP preempts SCs from 0 to 300 while for RP, the numbers of SCs preempted from the 8 eMBB users are 30, 30, 60, 30, 30, 30, 60, 30, respectively.

In Fig. 4.8a and Fig. 4.8b, we compare the sum eMBB utility ($\sum_n \log \bar{C}_n$) achieved by



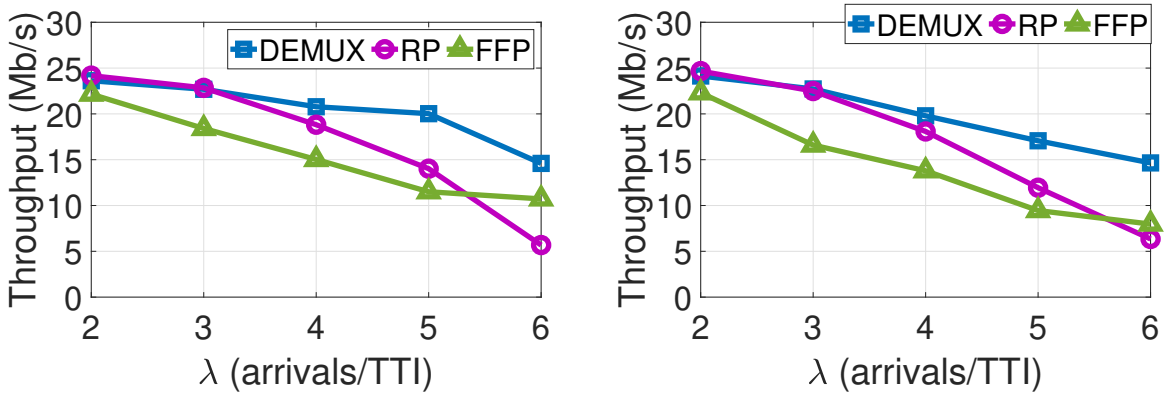
(a) CDL (b) TDL

Figure 4.8: Sum of utility comparison for 10 eMBB users under CDL and TDL channel models.

the three schemes under CDL and TDL channel fading models, respectively. As expected, the sum utility decreases as URLLC arrival rate increases under each scheme. Among the three schemes, DEMUX offers the best performance. When URLLC traffic load is low ($\lambda = 2$ and 3), the performance of DEMUX and RP is comparable but still better than FFP. The performance gap between DEMUX and the other two schemes widens when URLLC traffic load increases.

Fig. 4.9a and Fig. 4.9b show the eMBB cell throughput ($\sum_n \bar{C}_n$) achieved by the three schemes under CDL and TDL fading models, respectively. Again, DEMUX has the best performance among the three and the performance gap between DEMUX and the other two schemes widens as arrival rate increases. Specifically, the largest performance gains of DEMUX over FFP are 76% under CDL and 81% under TDL (when $\lambda = 5$), respectively. The largest gains of DEMUX over RP are 157% under CDL and 131% under TDL (when $\lambda = 6$), respectively.

(ii) 30 eMBB Users Next, we run experiments with 30 eMBB users under the same settings. Fig. 4.10a and Fig. 4.10b show the sum eMBB utility achieved by DEMUX, RP and FFP



(a) CDL (b) TDL

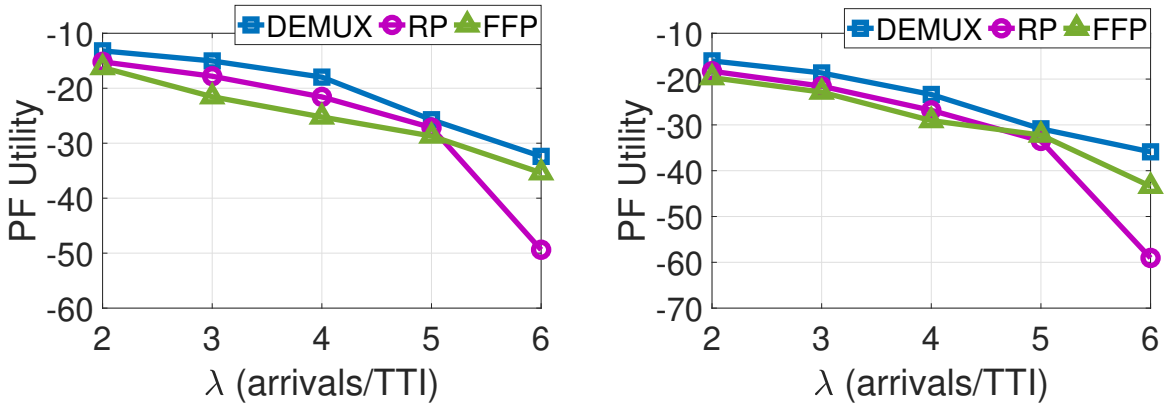
Figure 4.9: Cell throughput comparison for 10 eMBB users under CDL and TDL channel models.

under CDL and TDL fading models, respectively. We can see that DEMUX outperforms the other two schemes under all URLLC traffic loads and fading models.

Total cell throughput performance of the three schemes under CDL and TDL fading models is shown in Fig. 4.11a and Fig. 4.11b, respectively. Again, DEMUX achieves the highest throughput among the three. In particular, the largest gains of DEMUX over RP are 75% under CDL and 121% under TDL (when $\lambda = 6$), respectively. The largest gains of DEMUX over FFP are 27% under CDL and 21% under TDL (when $\lambda = 4$), respectively.

The fundamental reason that DEMUX can outperform RP and FFP is that DEMUX is designed to learn an optimal solution to the URLLC preemption problem, while RP and FFP are heuristics that one could only hope to find "good" solutions. Our results presented above demonstrate that DEMUX can successfully accomplish its goal and achieve superior performance than RP and FFP.

Computation Time To check whether or not DEMUX can meet the requirement for real-time scheduling, it is only necessary to examine its execution time on the scheduling plane, which consists of forward propagation time of neural network μ, and computation



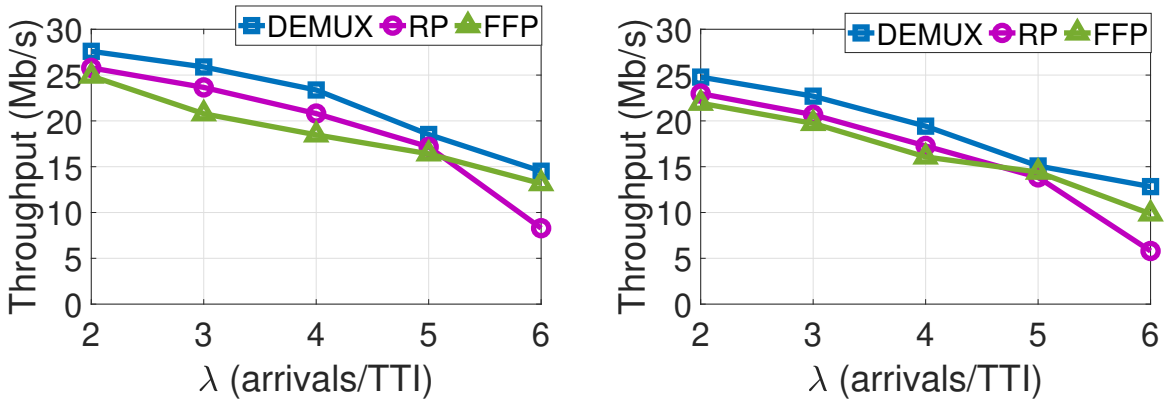
(a) CDL (b) TDL

Figure 4.10: Sum of utility comparison for 30 eMBB users under CDL and TDL channel models.

Table 4.4: Computation time of DEMUX. Results are averaged over 10^5 TTIs.

# of eMBB users	μ (ms)	$f \circ g$ (ms)	Total (ms)
10	0.276	0.088	0.364
30	0.269	0.092	0.361

time of post-processor g and translator f. The execution time of DEMUX is evaluated using an Intel Xeon E5-2687W v4 CPU and the results are shown in Table 4.4. First and foremost, we find that our CPU-based implementation of DEMUX meets the real-time requirement under NR Numerology 0 and 1, where time slot duration is 1 ms and 0.5 ms, respectively. For Numerology 2 and 3 with even shorter time slot duration (250 μs and 125 μs), one could employ GPU-based implementation (e.g., the cuDNN platform from NVIDIA [96]) to accelerate forward propagation of neural network μ. Second, we find that the computation time of DEMUX does not increase as the number of eMBB users grows. The reasons are that: (i) forward propagation time of μ only depends on the numbers of layers and neurons per layer in its structure; (ii) execution time of g and f is determined by the maximum number of eMBB users that can be scheduled per TTI (i.e., M^{RBG}) and is independent of



(a) CDL (b) TDL

Figure 4.11: Cell throughput comparison for 30 eMBB users under CDL and TDL channel models.

the total number of users in the cell.

4.10 Chapter Summary

In this chapter, we investigated dynamic eMBB/URLLC multiplexing via a preemptive puncturing mechanism that was proposed in 3GPP standards body. We studied an important optimization problem under this mechanism on how to spread SCs preempted by each URLLC transmission across eMBB users so that the adverse impact on eMBB is minimized. A major technical challenge is that this problem cannot be solved using traditional model-based optimization approaches. To address this challenge, we proposed DEMUX – a novel model-free DRL based solution. Key contributions in the design of DEMUX include: (i) DEMUX is able to learn an optimal algorithm for URLLC preemption using deep function approximators without the need for a prior explicit problem formulation. (ii) DEMUX is the first known design that employs DRL method with large and continuous action domain for enabling dynamic eMBB/URLLC multiplexing. (iii) To achieve fast and stable convergence of learning, we proposed to augment DDPG method by adapting the learning objective and

removing additional target networks based on intrinsic properties of the URLLC preemption problem. (iv) To ensure feasibility, we proposed a novel approach based on KL divergence to converting an unconstrained output of a neural network into a feasible URLLC preemption solution. (v) For implementation, we built an experiment platform consisting of a PHY-MAC NR simulator and a DRL learning module with a scalable object-oriented design that can be used for validating the performance of DEMUX and other benchmark schemes. (vi) Through extensive experiments using our platform, we showed that DEMUX significantly outperforms heuristic algorithms proposed in the 3GPP standards body and the literature (up to 130% performance gain in test network scenarios). Further, we showed that DEMUX meets the timing requirement for real-time scheduling in NR.

Chapter 5

A DL-based Link Adaptation for eMBB/URLLC Multiplexing in 5G NR

5.1 Introduction

5G NR has been developed with the objective of supporting services types with extremely diverse performance requirements under a unified radio interface [34]. Among the service types defined in NR, enhanced Mobile BroadBand (eMBB) and Ultra-Reliable and Low Latency Communications (URLLC) are two important categories that target at drastically different applications. While eMBB is expected to offer extremely high throughput for human-oriented services (e.g., 4K/8K video streaming and train/airplane communications) [81], URLLC has the goal of achieving an end-to-end latency on 1 ms time scale with a guarantee of ultra-reliability (e.g., with a $1 - 10^{-5}$ packet success probability) to meet the requirements of mission-critical applications (e.g., industrial IoT and autonomous driving) [82].

With such drastic differences in service requirements, eMBB and URLLC data need to be scheduled and transmitted on very different time scales. An eMBB transmission time

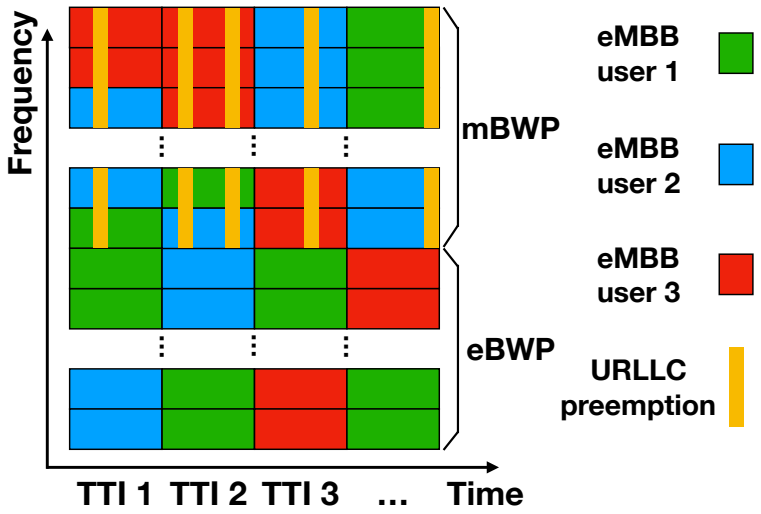


Figure 5.1: An illustration of URLLC preemptive puncturing.

interval (TTI) may span one or multiple time slots to achieve a high spectral efficiency, where a time slot consists of 14 OFDM symbols [2]. In contrast, a URLLC packet will be transmitted using a mini-slot in order to meet its latency requirement, where a mini-slot may contain as few as 2 OFDM symbols [35]. As a result, when eMBB and URLLC are dynamically multiplexed on the same channel, the allocation of radio resources becomes very complicated.

Since URLLC traffic is sporadic in nature, it would be inefficient to statically reserve a portion of the channel bandwidth for URLLC transmissions [84]. A more efficient approach is to allow URLLC packets to preempt ("puncture") resources from on-going eMBB transmissions. This novel mechanism, termed "preemptive puncturing", was proposed in 3GPP standards body [83]. With preemptive puncturing, each arrived URLLC packet is immediately scheduled for transmission for the following mini-slot using resources that have already been allocated to eMBB traffic. In the meantime, scheduled eMBB transmissions in those mini-slots will cease. An illustration of this mechanism is given in Fig. 5.1. For the portions of the eMBB transport blocks (TB) that are corrupted by the URLLC traffic, an indication signal is sent to each eMBB user to specify the time and frequency positions of

the punctured resources [89]. This information will be used by eMBB users to improve the LDPC decoding (e.g., for flushing out corrupted data bits).

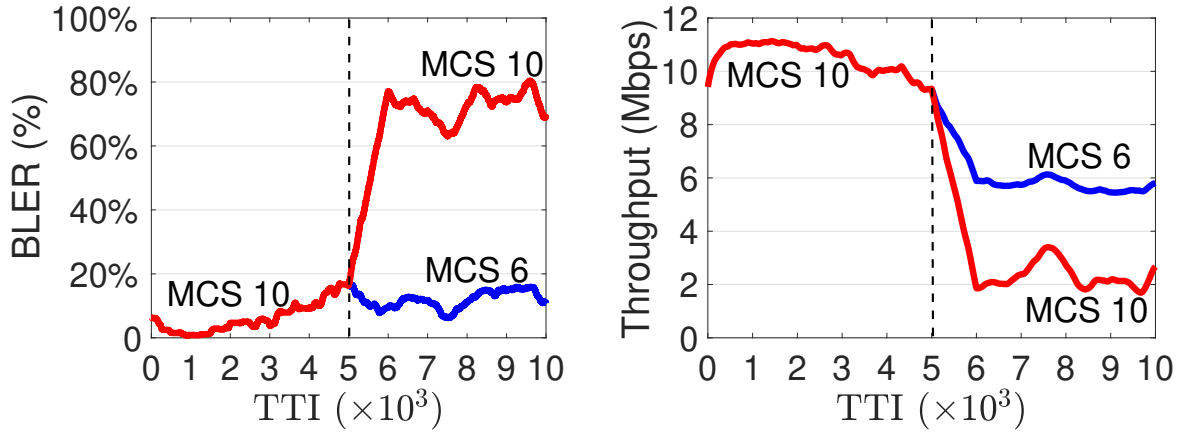
Although preemptive puncturing can effectively reduce the latency of URLLC, it will lead to a performance degradation for eMBB services in terms of data throughput and link reliability. Here, reliability is measured as the proportion of packets that are successfully decoded on an eMBB link. Specifically, when one or more URLLC packets were puncturing an eMBB transmission, the block-error rate (BLER) at the eMBB receiver increases and the decoding of the received transmission is more likely to fail. Such an increase of BLER means a deteriorated link reliability (i.e., 1-BLER) and will cause a loss of the achieved eMBB data throughput since each transmission has a higher probability to fail. To mitigate this adverse impact, a promising approach is to design a link adaptation (LA) mechanism for eMBB that is robust under URLLC puncturing. By LA, it means the dynamic adjustment of transmission parameters such as modulation type and code rate based on the instantaneous channel quality and interference on the eMBB link [47].

In this chapter, we explore this approach and focus on achieving LA through dynamic selection of modulation and coding scheme (MCS) [36] for each eMBB transmission corresponding to the given channel conditions and URLLC puncturing on its allocated resources. Specifically, there are 29 MCS levels defined in 5G standard [36], with a higher MCS level corresponding to a higher spectral efficiency (bits/s/Hz) but requiring a better channel quality. The problem we will study is to select one among the 29 MCSs for each eMBB transmission so as to maximize eMBB data throughput while guaranteeing the link reliability under the impact of URLLC puncturing.

A Motivating Example To understand how URLLC puncturing impacts the performance of eMBB and how LA can help mitigate such adverse impact, we conduct a link-level simulation study as follows. Consider an eMBB communication link between a base station

(BS) and an eMBB user (see Section 5.9 for parameter settings). In each TTI, we allocate a random number of resource blocks (RB) on the channel to this link to mimic RB allocation in a multi-user 5G cell. In Fig. 5.2, we show the BLER and throughput performance of this link under adaptive and non-adaptive MCS configurations before and after the occurrence of URLLC puncturing. Specifically, before the 5,000-th TTI, there is no URLLC traffic and the MCS is set to level 10 (QPSK modulation and a 0.6631 LDPC code rate). As shown in Fig. 5.2a and 5.2b, the average BLER of the link is 6.2% and the average eMBB throughput is 10.41 Mbps before the 5,000-th TTI. Starting from the 5,000-th TTI, we generate URLLC traffic following a Poisson process with an arrival rate of 3 packets/TTI. After the 5,000-th TTI, under a non-adaptive MCS configuration (still using MCS 10 for this link), the average BLER increases significantly to 67.6% and the average throughput drops to 2.36 Mbps. However, if an adaptive MCS configuration is employed, e.g., MCS is adjusted to level 6 (QPSK modulation and a 0.3701 code rate) after the 5,000-th TTI, the average BLER becomes 12.0% and the average eMBB throughput is 5.66 Mbps. Comparing the results in Fig. 5.2, it is clear that MCS selection is critical to mitigate the impact of URLLC puncturing on eMBB performance.

Although URLLC's adverse impact on eMBB is similar to the interference between communication links in 4G LTE, existing LA methods designed for LTE cannot be applied to address this problem. This is because the behavior of URLLC puncturing in 5G is fundamentally different from that of the interference effects in LTE networks (e.g., the inter-cell interference). Specifically, in LTE, LA is implemented as the adaptive modulation and coding (AMC) component [47, 108]. Under AMC, the MCS for a transmission in a TTI is selected based on the user's channel quality indicator (CQI) report in the prior TTI. Since there is a strong temporal correlation of channel conditions across the TTIs (within the channel coherence time), CQI report for the previous TTI can still be quite useful for the



(a) BLER (b) Throughput

Figure 5.2: BLER and throughput performance of an eMBB link before and after the occurrence of URLLC puncturing under adaptive and non-adaptive MCS configurations.

current TTI [108]. But this is hardly true in the case of URLLC's interference to eMBB, as the arrival of URLLC traffic in each TTI is sporadic and lack of correlation. As a result, CQI report on URLLC puncturing for the previous TTI is hardly useful for the current TTI. In other words, LA approaches based on the correlation of channel conditions in consecutive TTIs are no longer useful for the eMBB MCS selection (LA) in the presence of URLLC puncturing. Thus our first objective is to develop a new LA approach that can address the impact of sporadic URLLC puncturing.

In addition to the inability to address the impact from URLLC puncturing, there also exists some well-known limitation with existing CQI-based LA methods. To see this, it is important to understand how CQI works. In a TTI, CQI is obtained by reducing a high-dimensional description of the channel conditions across a large number of sub-carriers (SC) to a one-dimensional metric through some non-linear function, e.g., sum of exponential functions in the well known exponential effective SNR mapping (EESM) method [109, 110, 111]. Such a mapping, although simple, suffers from significant information loss of the channel conditions, which leads to performance issues in MCS selection and BLER guarantee

[110, 111, 112]. It has been shown that CQI-based LA may lead to over an order of magnitude variation of BLER under different channel realizations [110, 111]. For this reason, our new LA approach will be no longer based on CQI.

Main Contributions The objectives of this chapter are twofold. First and foremost, we want to design an LA method that can maximize eMBB's throughput in the presence of URLLC puncturing, while guaranteeing a link's BLER target for reliability.[1] Second, in the design of this new LA scheme, we want to go beyond the existing CQI methods when handling the information of channel conditions and URLLC traffic. To achieve these objectives, we present DELUXE – a deep-learning-based (DE) link adaptation design (L) for URLLC's multiplexing with eMBB (UXE). The basic idea behind DELUXE is to exploit deep learning (DL) to make reliable and efficient MCS selection based on *both* the *short-term channel condition* information that has strong correlation within channel coherence time, *and* the *long-term URLLC traffic behavior* which does not exhibit any short-term correlation. Specifically, DELUXE uses a deep function approximator (DFA) to make better utilization of information regarding channel conditions and URLLC puncturing than the simple mapping functions employed by the CQI methods.

Although the use of DL for LA and MCS selection in wireless networks has been investigated in previous works (see, e.g., [113, 114]), none of them has addressed the impact of URLLC puncturing on MCS selection. Further, due to the uniqueness of the problem in this chapter, none of the existing solutions can be applied to address the problem in this chapter.

The main contributions of this chapter are summarized as follows:

- DELUXE presents the first successful LA design that can achieve reliable and efficient eMBB transmission in the presence of URLLC puncturing. In each TTI, DELUXE

[1] One reason for enforcing a BLER target is to avoid too frequent re-transmissions of eMBB TBs which will result in deteriorated latency performance for eMBB services [47].

finds the highest possible MCS level for each scheduled eMBB transmission while guaranteeing a given target BLER.

- DELUXE is also the first LA design for dynamic eMBB/URLLC multiplexing that exploits DL. Through the use of a DFA, DELUXE makes MCS selection for eMBB based on both the short-term channel condition information and the long-term URLLC traffic behavior. This capability mitigates the limitation of the existing CQI-based LA methods, which select MCS only based on the short-term channel conditions.

- We propose a mapping to translate the high-dimensional information of channel condition and URLLC puncturing on an eMBB transmission into a low-dimensional representation. Our method can effectively minimize the information loss caused by such translation. This mapping significantly reduces the dimension of input information and allows us to simplify the DFA's multi-layer structure and reduce its execution time (for forward propagation).

- We propose a learning method that allows a DFA to learn and predict the BLERs of an eMBB transmission under each MCS level. With this method, the DFA can be learned based on the decoding results from a large number of eMBB transmission samples, without the need to determine the actual BLER of each transmission sample. Moreover, the learned DFA can simultaneously predict the BLERs for all MCS levels through a single forward propagation. To mitigate the potential errors in the BLER predictions, we design a calibration mechanism to make a final tuning for the MCS selection by taking into consideration the recent decoding result history of each eMBB user.

- To ensure the total execution time can meet the real-time requirement in NR (under 100 μs), we employ a hybrid CPU and GPU based architecture to implement DELUXE.

In our implementation, the MCS selections for all eMBB transmissions scheduled in a TTI are performed in parallel through efficient matrix operations and single-instruction multiple-data (SIMD) processing.

- We evaluate the performance of DELUXE through an extensive link-level simulation study under practical network settings. Our results show that DELUXE can effectively maintain the target BLER for eMBB services under the impact of URLLC puncturing while the state-of-the-art EESM method fails to do so (by a wide margin). Moreover, DELUXE achieves eMBB throughput that is comparable to that with EESM, which is considered the most spectral efficient LA method for 4G LTE and 5G NR. Finally, our GPU-based implementation of DELUXE can successfully meet the real-time requirement in 5G NR (under 100 μs).

The rest of the chapter is organized as follows. In Section 5.2, we review related work on dynamic eMBB/URLLC multiplexing and the applications of DL in wireless communications and networking. In Section 5.3, we describe the system model and the problem that we study in this chapter. Section 5.4 presents the architecture and key ideas of DELUXE. Through Section 5.8 to 5.7, we present each of the three stages of DELUXE. In Section 5.8, we describe our GPU-based real-time implementation for DELUXE. Section 5.9 presents our performance evaluation of DELUXE. Section 5.10 concludes this chapter.

5.2 Related Work

In this section, we review related work in two thrusts that are most relevant to this research: (i) multiplexing of eMBB and URLLC (on the same channel), and (ii) the use of DL in wireless communications and networking.

5.2.1 eMBB/URLLC Multiplexing

Related work on dynamic multiplexing of eMBB and URLLC based on preemptive puncturing includes [84, 86, 92, 118, 119]. In [84], the authors conducted system-level simulations and confirmed that preemptive puncturing achieves significantly higher spectral efficiency than static bandwidth division under sporadic URLLC packet arrivals. In [86], the authors studied a bandwidth allocation problem for eMBB transmissions under linear and nonlinear models for the impact of URLLC puncturing. In [92], the authors studied RB allocation for eMBB and the scheduling of URLLC puncturing. In [118], the authors applied deep reinforcement learning (DRL) to dynamic eMBB/URLLC multiplexing, with the goal of minimizing the adverse impact of URLLC traffic on eMBB services. In [84, 86, 92, 118], the problem of MCS selection for eMBB under the impact of URLLC puncturing was not considered.

MCS selection for eMBB was considered in [119]. In this work, the authors used system-level simulations to evaluate the impact of preemptive URLLC puncturing on eMBB. The well-known outer loop link adaptation (OLLA) algorithm was employed for MCS selection (for eMBB). However, their results showed that eMBB's BLER target could not be satisfied when URLLC puncturing is present.

5.2.2 DL for Wireless Communications and Networking

DL and DRL methods have been applied to address wireless communications and networking problems extensively in recent years (for a survey, see [124]). This line of research has two main branches. In the first branch, the goal is to custom-design a neural network structure for a specific problem (see, e.g., [120, 121, 122]). Due to problem-specific nature, a custom-designed neural network structure, in general, cannot be easily extended to solve other

problems, such as the MCS selection problem in this chapter.

In the second branch, a problem under study is first cast into a DRL framework and then solved by designing a suitable neural network structure (see, e.g., [100, 102, 123]). The most widely used DRL framework in wireless communications and networking is deep Q-learning (DQL) [116]. Unfortunately, DQL-based approaches are not suitable for solving the MCS selection problem in this chapter. This is because Q-learning is a greedy algorithm and therefore, it cannot handle the BLER constraints and ensure feasibility in the decision making process.

5.3 System Model and Problem Statement

System Model Table 5.1 lists the notation used in this chapter. Consider a 5G NR macro-cell with its base station (BS) located at the cell center and a number of eMBB and URLLC users within the cell's coverage, as shown in Fig. 5.3. Denote $\mathcal{X}$ as the set of eMBB users in the cell. In the downlink direction, the data traffic of eMBB and URLLC is dynamically multiplexed on the same channel. To focus on the impact of URLLC traffic on eMBB transmission, we assume a full-buffer model for each eMBB user $x \in \mathcal{X}$, i.e., the eMBB buffers at the BS are always nonempty in the downlink direction.

For eMBB transmission, time domain is divided into consecutive *transmission time intervals* (TTIs). Scheduling for eMBB transmission is performed for each TTI. Denote t as the time index of a TTI. Typically, a TTI may consist of one or more time slots, with each time slot consisting of 14 OFDM symbols [2]. Denote F as the number of OFDM symbols in a TTI (which is a multiple of 14). Assume that under the fast fading effects, the channel condition for an eMBB link (from the BS to an eMBB user) varies in each TTI.

Table 5.1: Notation in Chapter 5

Symbol	Definition
$a_{i,m}(t)$	The binary decoding result for $e_i(t)$ under MCS m
$\mathcal{B}$	The set of all RBs on the operating channel
$\mathcal{B}_i(t)$	The set of RBs allocated to $e_i(t)$
$\boldsymbol{c}_i(t)$	The vector describing the RE-level channel conditions for $e_i(t)$
$\mathcal{E}(t)$	The set of eMBB transmissions scheduled for TTI t
$e_i(t)$	The i-th eMBB transmission in $\mathcal{E}(t)$
F	The number of OFDM symbols in a TTI
I	The number of OFDM symbols in a mini-slot
J	The total number of iterations in the learning phase of Algorithm 5
L	The number of elements in $\boldsymbol{\xi}$
$\mathcal{M}$	The set of MCS levels that can be used for eMBB
M	$= \lvert\mathcal{M}\rvert$, the number of MCS levels in $\mathcal{M}$
$\boldsymbol{p}_i(t)$	The vector specifying the RE-level URLLC puncturing on $e_i(t)$
$\boldsymbol{q}_i(t)$	The vector of $(1 - r_{i,m}(t))$ values under each MCS m for $e_i(t)$
$r_{i,m}(t)$	The BLER for $e_i(t)$ under MCS m
$\boldsymbol{s}_i(t)$	The original input vector for $e_i(t)$
$\tilde{\boldsymbol{s}}_i(t)$	The low-dimensional representation of $\boldsymbol{s}_i(t)$
S	The number of SCs in an RB
T	The total number of TTIs in the data collection phase of Algorithm 5
$\mathcal{X}$	The set of eMBB users in the cell
Z	The update period for β^x in number of eMBB transmissions
β^x	The calibration parameter for eMBB user $x \in \mathcal{X}$
γ	The BLER target for eMBB links
γ_0	The cushion parameter in Algorithm 6
Δ_{D}	The step size for decreasing β^x in Algorithm 6
Δ_{I}	The step size for increasing β^x in Algorithm 6
Θ	The set of trainable parameters in μ's multi-layer structure
λ	The URLLC packet arrival rate
μ	The DFA in the Stage II of DELUXE
$\mu\left(\tilde{\boldsymbol{s}}_i(t)\lvert\Theta\right)$	The output vector of μ given the input $\tilde{\boldsymbol{s}}_i(t)$ and parameters Θ
$\mu_m\left(\tilde{\boldsymbol{s}}_i(t)\lvert\Theta\right)$	The m-th output entry in $\mu\left(\tilde{\boldsymbol{s}}_i(t)\lvert\Theta\right)$
$\boldsymbol{\xi}$	The vector used for generating the low-dimensional input representation $\tilde{\boldsymbol{s}}_i(t)$
ρ	The step size in the learning phase of Algorithm 5
$\upsilon(e_i(t))$	$\in \mathcal{X}$, the receiving eMBB user corresponding to $e_i(t)$

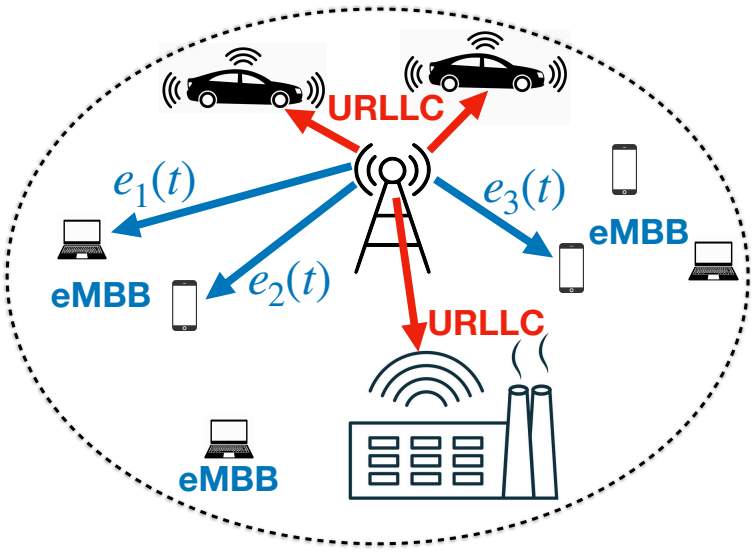


Figure 5.3: An NR macro-cell with eMBB/URLLC multiplexing.

In the frequency domain, the bandwidth of the operating channel is divided into a large number of sub-carriers (SC). Assume that under frequency-selective channel fading, the channel condition for an eMBB link varies on each SC. A contiguous S SCs are grouped into an RB. According to NR standard [2], $S = 12$. An RB is the minimum frequency unit for resource scheduling over the duration of a TTI. Denote $\mathcal{B}$ as the set of all RBs on the channel. For each TTI t, the RB allocation for eMBB transmissions is determined by the BS scheduler in TTI $(t-1)$ following a scheduling algorithm (e.g., proportional-fair (PF) algorithm [52, 55]). Let $\mathcal{E}(t)$ denote the set of eMBB transmissions scheduled for TTI t, where each eMBB transmission is a communication link from the BS to an eMBB user $x \in \mathcal{X}$. For example, $\mathcal{E}(t)$ in Fig. 5.3 consists of 3 eMBB transmissions: BS to $e_1(t)$, $e_2(t)$, and $e_3(t)$, respectively, where $e_i(t)$ denotes the i-th transmission in $\mathcal{E}(t)$. Denote $v(e_i(t)) \in \mathcal{X}$ as the receiving eMBB user corresponding to $e_i(t)$. Denote the set of RBs allocated to transmission $e_i(t)$ as $\mathcal{B}_i(t)$.

Given a scheduled eMBB transmission $\mathcal{E}(t)$ for a given TTI t, one needs to select an MCS for each $e_i(t) \in \mathcal{E}(t)$. Denote $\mathcal{M} = \{1, 2, \cdots, 29\}$ as the set of 29 MCS levels defined

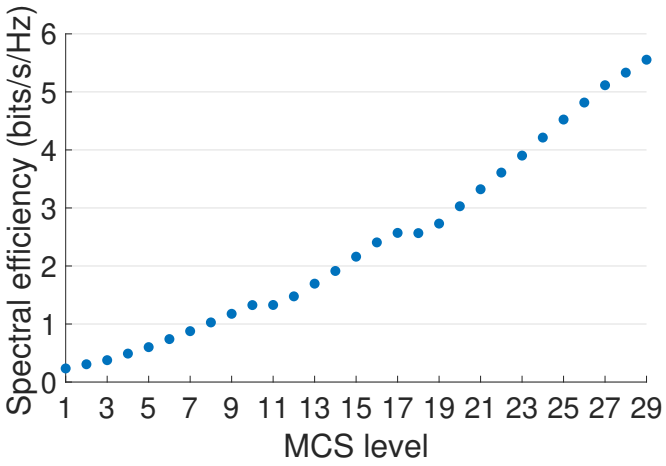


Figure 5.4: Spectral efficiencies of the 29 MCS levels defined in NR standard (Table 5.1.3.1-1 in [36]).

in the NR standards [36]. Each MCS represents a combination of a specific modulation type (QPSK, 16-QAM, or 64-QAM) and LDPC code rate (between 0 and 1). The spectral efficiency of each MCS level is shown in Fig. 5.4. A key constraint associated with MCS is that the same MCS must be used for all the RBs allocated to an eMBB transmission [36]. The main reason behind this constraint is that the benefit of using different MCS levels for different RBs (in terms of throughput gain) cannot justify the excessive control signaling overhead associated with it [47].

For URLLC traffic, we assume that URLLC packets arrive at the BS sporadically following a Poisson process with an arrival rate λ (in number of packets per TTI). Different from eMBB, each URLLC packet is transmitted using a "mini-slot", where a mini-slot contains a much fewer number of OFDM symbols (e.g., 2, 4, or 6 [2]) than a regular time slot (14 symbols). Denote I as the number of OFDM symbols in a mini-slot. A URLLC transmission is scheduled immediately for the next mini-slot following the packet arrival through preemptive puncturing as described in Section 5.1. Specifically, in the frequency domain, a set of contiguous RBs starting from the highest (or lowest) frequency will be punctured in the scheduled mini-slot (see Fig. 5.1).

Following NR standard [2], we consider a bandwidth part (BWP) based channel division. Under this bandwidth division, the operating channel is divided into two non-overlapping frequency parts (or BWPs): the multiplexing BWP (mBWP) and the eMBB exclusive BWP (eBWP), respectively. As shown in Fig. 5.1, in mBWP, the multiplexing of eMBB and URLLC occurs through preemptive puncturing. In eBWP, the bandwidth is used exclusively for eMBB transmissions. Benefits of such a partition scheme include low control channel overhead for indicating positions of URLLC puncturing and the compatibility with current control channel design in NR standards [89]. In each TTI t, an eMBB transmission $e_i(t)$ may be allocated with RBs from either mBWP, or eBWP, or both. Such an RB allocation decision is done by the eMBB scheduler at the BS and is beyond the scope of this chapter.

Problem Statement The problem we want to solve in this chapter is to find an optimal MCS level $m_i^*(t)$ for each $e_i(t) \in \mathcal{E}(t)$ so as to maximize throughput while satisfying a given BLER target. Such an MCS selection for each $e_i(t) \in \mathcal{E}(t)$ is based on the following input: i) the code block (CB) length, ii) the channel conditions, and iii) URLLC puncturing on the RBs allocated to $e_i(t)$. More discussion about these three inputs will be given in Section 5.5.1.

Denote γ as the BLER target for eMBB. For 5G, γ is typically set to 10% [126], meaning that the decoding failure probability of an eMBB transmission should not exceed 10%. Then our problem can be stated mathematically as follows:

$$m_i^*(t) = \underset{m \in \mathcal{M}}{\arg\max}\ \mathrm{SE}(m),\ \text{subject to}: r_{i,m}(t) \leq \gamma, \tag{5.1}$$

where $\arg\max$ represents the variable value that maximizes the objective function, $\mathrm{SE}(m)$ is the spectral efficiency for MCS level m (see Fig. 5.4), and $r_{i,m}(t)$ denotes the BLER for $e_i(t)$ when MCS is set to m.

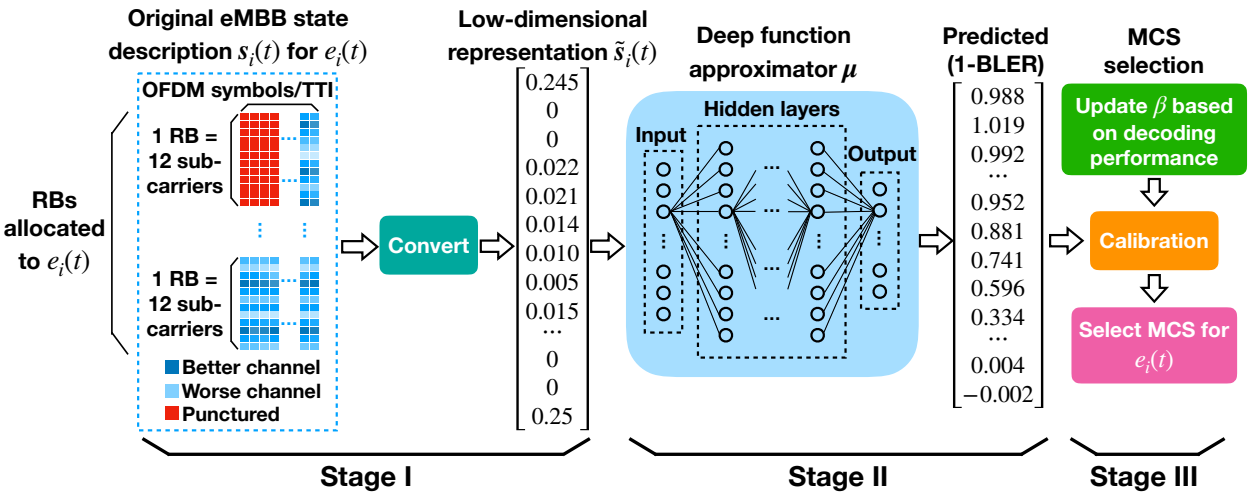


Figure 5.5: Architectural overview of DELUXE. This figure illustrates the MCS selection process for a given $e_i(t)$.

Technical Challenges A major challenge in solving the MCS selection problem in (5.1) is that it is analytically intractable to determine the BLER $r_{i,m}(t)$ for each $e_i(t)$ under a given MCS m. This is because 5G NR employs LDPC as the FEC channel code for eMBB transmissions [94]. It is well-known that LDPC decoding performance (in terms of BLER) cannot be modeled using a closed-form mathematical expression. That is, we cannot determine the $r_{i,m}(t)$ values for each $e_i(t)$ even when the perfect information of channel condition and URLLC puncturing on $e_i(t)$ is available. As a result, the problem in (5.1) cannot be formulated as a closed-form optimization problem where conventional optimization techniques may be applied.

Another technical challenge associated with the problem in (5.1) is that its solution must be obtained in real time, i.e., an optimal MCS must be selected for each $e_i(t) \in \mathcal{E}(t)$ within a TTI's duration. In 5G NR, the shortest TTI duration for eMBB is 125 μs under Numerology 3 [2]. This means that the MCS selection problem in (5.1) must be solved on $\sim$100 μs time scale. Based on our past experience, one must pursue parallel processing in implementation to meet such a stringent real-time requirement.

5.4 DELUXE: Architecture and Key Ideas

Given the set of eMBB transmissions $\mathcal{E}(t)$ scheduled for a TTI t, the objective of DELUXE is to determine the optimal MCS $m_i^*(t)$ for each $e_i(t) \in \mathcal{E}(t)$ as described in problem (5.1). For each $e_i(t)$, we need to provide the following information to DELUXE: i) the length of the eMBB CB in $e_i(t)$, ii) channel conditions on the RBs allocated to $e_i(t)$, and iii) URLLC puncturing on the RBs allocated to $e_i(t)$. The basic idea behind DELUXE is to employ a DFA (a feedforward neural network), denoted by μ, to predict the value of $(1 - r_{i,m}(t))$ for a given $e_i(t)$ under each MCS $m \in \mathcal{M}$. Then based on the predicted $(1 - r_{i,m}(t))$ values, we will be able to find an optimal MCS that solves problem (5.1).

Fig. 5.5 shows the overall architecture of DELUXE. There are three stages in DELUXE, i.e., Stage I, II and III. Next, we briefly describe the key ideas in each stage. Technical details of the three stages will be given in Section 5.5, 5.6 and 5.7, respectively.

Stage I: *Reducing complexity of input data.* If one used the original high-dimensional information for each $e_i(t)$ as an input to μ, a very complex hidden-layer structure in μ would be necessary (e.g., with a series of convolutional, pooling and fully-connected layers) to extract the essential information to predict $(1 - r_{i,m}(t))$ values. This will lead to very high processing/inference time of μ that is far beyond our stringent real-time requirement ($\sim$100 μs). To address this issue, in Section 5.5, we propose an approach to transform the original high-dimensional input information for each $e_i(t)$ into a low-dimensional representation with a minimized information loss.

Stage II: *Learning and prediction.* The design and learning of DFA μ is the core component of DELUXE. The input to μ is a low-dimensional representation of the channel condition and URLLC puncturing information for $e_i(t)$ (obtained in Stage I). The output of μ is a real-valued vector of length $M = |\mathcal{M}|$, with each entry representing the

predicted $(1-r_{i,m}(t))$ for $e_i(t)$ under each MCS $m \in \mathcal{M}$. We employ a fully-connected layer structure for μ (i.e., a neuron within a layer is fully connected to all neurons in the next layer) such that it can offer a strong universal function approximating capability that can meet our need [95]. In Section 5.6, we will present the details of our learning method for μ based on the decoding results from a large number of eMBB transmission samples.

Stage III: *Calibration and MCS selection.* Our proposed learning method allows μ to predict the $(1 - r_{i,m}(t))$ values for a given $e_i(t)$ under each MCS m. However, due to the nature of a DFA, there may exist errors in these predictions. To mitigate the impact of such errors in MCS selection, in Section 5.7, we propose a method to dynamically calibrate the $(1 - r_{i,m}(t))$ predictions from μ for each eMBB user based on its recent history of actual decoding results.

5.5 Reducing Complexity of Input Data

In this section, we describe the operation in Stage I (see Fig. 5.5), which is to transform the original high-dimensional input data for an eMBB transmission $e_i(t)$ into a low-dimensional representation. We first describe the scope of the input data (Section 5.5.1). Then we propose a method to achieve such transformation (Section 5.5.2).

5.5.1 Input Data

In each TTI t, we can use a vector $\boldsymbol{s}_i(t)$ to describe an eMBB transmission $e_i(t) \in \mathcal{E}(t)$. $\boldsymbol{s}_i(t)$ contains the following information: (i) the CB length in $e_i(t)$, (ii) the URLLC puncturing on $e_i(t)$, and (iii) the channel conditions (across both frequency and time) on $e_i(t)$. The BLER

Table 5.2: Determine the CB length for $e_i(t)$.

MCS	0-9	10-16	17-28
Modulation	QPSK	16-QAM	64-QAM
Bits/RE	2	4	6
Bits/RB	336	672	1,008
CB length	$336\|\mathcal{B}_i(t)\|$	$672\|\mathcal{B}_i(t)\|$	$1{,}008\|\mathcal{B}_i(t)\|$

of $e_i(t)$ is determined by these three types of input information. We now discuss each of these inputs in details.

(i) $\boldsymbol{s}_i(t)$'s first component describes the CB length of $e_i(t)$. Given the RB allocation $\mathcal{B}_i(t)$, we have the following question: what is the CB length under each of the 29 MCSs?

We can answer this question by constructing Table 5.2. In this table, each of the $M = 29$ MCSs corresponds to a specific modulation type (first row to second row). Under each modulation type, we know exactly how many codeword bits are carried in a resource element (RE),[2] as shown in the third row in the table. Since the number of REs contained in an RB within a TTI is a constant number (i.e., 14 OFDM symbols/TTI $\times$ 12 SCs/RB), the number of codeword bits per RB under each modulation type can be determined (fourth row in the table). This means that the CB length for $e_i(t)$ under each MCS is equal to $|\mathcal{B}_i(t)|$ multiplied by a constant, as shown in the last row of the table. In other words, for each of the 29 MCSs, we can find the CB length through Table 5.2 if we know $|\mathcal{B}_i(t)|$.

(ii) $\boldsymbol{s}_i(t)$'s second component is a vector $\boldsymbol{p}_i(t)$ that describes the URLLC puncturing on $e_i(t)$. Since puncturing occurs on the OFDM symbol level (in mini-slots), $\boldsymbol{p}_i(t)$ should have a granularity of an RE. In other words, the number of elements in $\boldsymbol{p}_i(t)$ should be equal to the total number of REs allocated to $e_i(t)$. Specifically, each element in $\boldsymbol{p}_i(t)$ indicates whether or not an RE is punctured by URLLC. Given RB allocation $\mathcal{B}_i(t)$, the total number

[2] An RE is defined as one OFDM symbol time duration by one SC in frequency [2].

of REs in $e_i(t)$ is $|\mathcal{B}_i(t)| \cdot S \cdot F$. Thus we have $|\boldsymbol{p}_i(t)| = |\mathcal{B}_i(t)| \cdot S \cdot F$.

However, since the URLLC packet arrivals are sporadic, we do not know how many packets will actually arrive in TTI t when generating the input vector $\boldsymbol{s}_i(t)$ (in TTI $t-1$). A reasonable estimate for the number of URLLC packets arrived in each TTI is λ, the Poisson arrival rate of URLLC packets. Furthermore, when generating $\boldsymbol{s}_i(t)$, we assume that the puncturing of the (expected arrival) URLLC packets occurs in consecutive mini-slots starting from the first in each TTI (within the mBWP). This assumption on puncture position can be justified by noticing that the duration of a TTI is shorter than the channel coherence time, meaning that the channel conditions only experience slight variation across the mini-slots within a TTI. Further, the impact of URLLC puncturing on specific mini-slots in a TTI is spread out uniformly across the eMBB CB by the bit interleaving operation [36].

(iii) The third component of $\boldsymbol{s}_i(t)$ is a vector $\boldsymbol{c}_i(t)$ to describe channel conditions on $e_i(t)$. Similar to $\boldsymbol{p}_i(t)$, $\boldsymbol{c}_i(t)$ should have a granularity at RE level, i.e., the number of elements in $\boldsymbol{s}_i(t)$ should be equal to the total number of REs in $e_i(t)$. That is, $|\boldsymbol{c}_i(t)| = |\mathcal{B}_i(t)| \cdot S \cdot F$.

For each element in $\boldsymbol{c}_i(t)$, we employ *post-detection-SNR* (PD-SNR) as the metric for channel condition. To concretize our discussion, we assume single-user single-layer MIMO is used for eMBB transmission.[3] Following frequency-major order in the frequency-time RE grid, consider the j-th RE allocated to $e_i(t)$. This corresponds to the j-th entry in $\boldsymbol{c}_i(t)$, which we denote as $c_{i,j}(t)$. Let $\boldsymbol{H}$ and $\boldsymbol{w}$ denote the channel matrix and the precoding vector on this RE. Then with MMSE detector, the PD-SNR (in dB) for this RE is [115]:

$$c_{i,j}(t) = 10 \log_{10} \left[(\boldsymbol{H}\boldsymbol{w})' \boldsymbol{H}\boldsymbol{w} \cdot v_0 \right], \tag{5.2}$$

where v_0 is the ratio of per-RE signal power over the noise power.

[3]The same approach described here can be applied to other MIMO transmission modes and detectors.

In summary, the input vector $\boldsymbol{s}_i(t)$ is written as:

$$\boldsymbol{s}_i(t) = [|\mathcal{B}_i(t)|;\ \boldsymbol{c}_i(t);\ \boldsymbol{p}_i(t)]\,. \tag{5.3}$$

There are two issues with $\boldsymbol{s}_i(t)$ in (5.3). First, the number of RBs $|\mathcal{B}_i(t)|$ allocated to $e_i(t)$ typically varies with TTI t [70]. As a result, the sizes of $\boldsymbol{c}_i(t)$ and $\boldsymbol{p}_i(t)$ both vary with TTI (as both depend on $|\mathcal{B}_i(t)|$). Hence, the dimension of $\boldsymbol{s}_i(t)$ varies in each TTI. Such variability in size makes $\boldsymbol{s}_i(t)$ unsuitable as an input to the DFA μ, which requires a constant input size that is independent of time. Second, the dimension of $\boldsymbol{s}_i(t)$ can be very large. For example, suppose an eMBB transmission $e_i(t)$ is allocated with 20 RBs. Then based on (5.3), we have $|\boldsymbol{s}_i(t)| = 6,720$ (when a TTI consists of only one time slot). With such a large-sized input, μ needs to have a large and complex layer structure to perform its function (extract the essential information from $\boldsymbol{s}_i(t)$). The inference time under such a large and complex structure will almost certainly violate our real-time requirement ($\sim$100 μs).

5.5.2 Transformation to Lower-Dimensional Representation

In this section, we propose a method to transform the original input $\boldsymbol{s}_i(t) = [|\mathcal{B}_i(t)|;\ \boldsymbol{c}_i(t);\ \boldsymbol{p}_i(t)]$ into a lower dimensional representation, denoted by $\tilde{\boldsymbol{s}}_i(t)$, for a given $e_i(t)$. First, we note that $|\mathcal{B}_i(t)|$ is a scalar and there is no room to compress it further. So our focus is to transform vectors $\boldsymbol{c}_i(t)$ and $\boldsymbol{p}_i(t)$ in $\boldsymbol{s}_i(t)$ into their low-dimensional representations, which are denoted by $\tilde{\boldsymbol{c}}_i(t)$ and $\tilde{\boldsymbol{p}}_i(t)$, respectively. Our objective for this transformation is three-fold:

(i) the dimension of $\tilde{\boldsymbol{s}}_i(t)$ must be constant and independent of t;

(ii) $\tilde{\boldsymbol{s}}_i(t)$ must have a much lower dimension compared to $\boldsymbol{s}_i(t)$;

(iii) the loss of information in the transformation should be minimized.

Key Idea The basic idea is to change the information granularity from RE level to something bigger (coarser). In the original input vector $\boldsymbol{s}_i(t)$, we have $\boldsymbol{c}_i(t)$ and $\boldsymbol{p}_i(t)$, both of which have a granularity at the RE level, leading to very large sizes for both vectors. Instead of RE level representation, we propose the following new representation for channel condition. We will use a set of intervals (orders of magnitude fewer than the number of REs) to record the number of entries in $\boldsymbol{c}_i(t)$ and $\boldsymbol{p}_i(t)$ that fall within each interval. Then we use these low-dimensional statistics as $\tilde{\boldsymbol{c}}_i(t)$ and $\tilde{\boldsymbol{p}}_i(t)$ to replace $\boldsymbol{c}_i(t)$ and $\boldsymbol{p}_i(t)$.

One would immediately notice that the above approach not only reduces information granularity, but also loses position information for the channel condition. So the question is: what is the theoretical basis for such a transformation? The answer lies in the bit interleaving operation during PHY-layer signal processing. Recall that after this operation, the codeword bits carried in the REs of $e_i(t)$ are uniformly spread out across the eMBB CB and no longer maintain their positions in the frequency-time RE grid. As a result, the RE position information carried in $\boldsymbol{c}_i(t)$ and $\boldsymbol{p}_i(t)$ is no longer useful and does not need to be maintained.

Such a transformation will meet the first two objectives for $\tilde{\boldsymbol{s}}_i(t)$ (constant size, smaller dimension). In the rest of this section, we will show the details of this transformation and address the third objective (to minimize information loss).

Transformation of $\boldsymbol{c}_i(t)$ We first discuss $\boldsymbol{c}_i(t)$. The property of bit interleaving operation allows us to turn away from maintaining per-RE based channel condition information. From (5.2), we see that the range of PD-SNRs (dB) is $(-\infty, +\infty)$. To guarantee a constant and finite size of the representation $\tilde{\boldsymbol{c}}_i(t)$, we propose to discretize the interval $(-\infty, +\infty)$ into a finite (constant) number of intervals and use the percentages of PD-SNRs in $\boldsymbol{c}_i(t)$ that fall into these intervals to characterize channel condition for eMBB $e_i(t)$.

Specifically, we employ a vector $\boldsymbol{\xi} = [\xi_1, \xi_2, \cdots, \xi_L]$ to divide $(-\infty, +\infty)$ into $(L+1)$ intervals: $(-\infty, \xi_1]$, $(\xi_1, \xi_2]$, $\cdots$, $(\xi_L, +\infty)$.[4] Then the representation $\tilde{\boldsymbol{c}}_i(t)$ is a vector of size $(L+1)$. That is,

$$\tilde{\boldsymbol{c}}_i(t) = [\tilde{c}_{i,1}(t), \tilde{c}_{i,2}(t), \cdots, \tilde{c}_{i,L+1}(t)], \tag{5.4}$$

where $\tilde{c}_{i,l}(t)$ is the percentage of PD-SNRs from the per-RE based $\boldsymbol{c}_i(t)$ that fall in the l-th interval. More precisely, $\tilde{c}_{i,l}(t)$ is the percentage of PD-SNRs on REs that are *not* punctured by URLLC.

A key question in this discretization process is the following: *For a given L, how to optimize ξ_1, ξ_2, $\cdots$, ξ_L in $\boldsymbol{\xi}$ so that the information loss in this transformation can be minimized?* In the following paragraphs, we address this important question.

Minimizing Information Loss We address the above question (minimizing information loss) by formulating an optimization problem as follows. For the objective function, we use the *entropy* [127] of PD-SNR as the metric to quantify the information loss due to our discretization of $(-\infty, +\infty)$ into $(L+1)$ intervals. Here the PD-SNR in (5.2) is a random variable for all possible eMBB transmissions in the cell with an *unknown* probability density function (PDF) over $(-\infty, +\infty)$.

Denote $H(l, l+1)$ as the PD-SNR's entropy within $(\xi_l, \xi_{l+1}]$. Further, denote $P(l, l+1)$ as the probability that the PD-SNR falls within $(\xi_l, \xi_{l+1}]$. Then our optimization problem

[4]The choice of L depends on the following two considerations: i) L should be small enough (orders or magnitude smaller than $|\boldsymbol{s}_i(t)|$) to meet the 5G real-time requirement; and ii) L should still be large enough so that $\tilde{\boldsymbol{s}}_i(t)$ contains enough information for an optimal MCS selection. In our experimental study, we found that $L = 100$ achieves a good balance between these two considerations.

can be formulated as:

$$\textbf{OPT-L:}$$
$$\text{minimize} \quad \sum_{l=1}^{L-1} P\left(l, l+1\right) H(l, l+1) \ ,$$
$$\text{subject to} \quad \xi_1 < \xi_2 < \cdots < \xi_L \ .$$

There are two problems with OPT-L. First, the PDF of PD-SNR is unknown. As a result, the probabilities $P\left(l, l+1\right)$ are unknown as well. Second, entropy $H(l, l+1)$, which is a function of the PD-SNR's PDF, cannot be determined in closed-form. To address the problems with $P\left(l, l+1\right)$ and $H(l, l+1)$ in OPT-L, we propose the following solutions.

Despite that the PDF of PD-SNR is unknown, we can still estimate its distribution through experiments and empirical data. Specifically, we can collect a set $\mathcal{N}$ of PD-SNR samples from all active eMBB links in the cell. Each sample element in $\mathcal{N}$ is the PD-SNR on an RE in an eMBB transmission. Let N denote the number of samples in $\mathcal{N}$ (e.g., $N = 10^6$). Then for an interval $(\xi_l, \xi_{l+1}]$, $P\left(l, l+1\right)$ can be estimated by calculating the proportion of samples in $\mathcal{N}$ that fall in this interval.

Although the above experiments can give us an estimate for $P\left(l, l+1\right)$, we still do not have a closed-form PDF for PD-SNR. So for entropy $H(l, l+1)$, we can obtain its upper bound by using a uniform distribution for the PDF of PD-SNR within each interval $(\xi_l, \xi_{l+1}]$. Then we have [127]

$$H(l, l+1) \leq \log\left(\xi_{l+1} - \xi_l\right) \ .$$

We will use $\log\left(\xi_{l+1} - \xi_l\right)$ as an approximation for $H(l, l+1)$ in the objective function of OPT-L.

With the above estimation for $P\left(l, l+1\right)$ and upper bound for $H(l, l+1)$ in OPT-L, we

propose a greedy algorithm to solve optimal $\boldsymbol{\xi} = [\xi_1, \xi_2, \cdots, \xi_L]$ to divide $(-\infty, +\infty)$ into $(L+1)$ intervals as follows.

(i) First, for the N samples from the experiment, we initialize vector $\boldsymbol{\xi}$ such that each of the $(L+1)$ intervals contains the same number of samples, i.e., $N/(L+1)$ samples in each of the intervals $(-\infty, \xi_1]$, $(\xi_1, \xi_2]$, $\cdots$, $(\xi_L, +\infty)$.

(ii) Next, we iteratively update one ξ_l $(l = 2, 3, \cdots, L-1)$ in $\boldsymbol{\xi}$ that offers the greatest reduction in the objective value in OPT-L for K iterations (or until no improvement is possible). Specifically, in each iteration, we slightly perturb each ξ_l $(l = 2, 3, \cdots, L-1)$ to both the left and right for a certain amount (e.g., such an amount may correspond to an increase or decrease of some fixed number of samples for the underlying intervals). For each perturbation, we calculate the change of objective value in OPT-L. Among the $2(L-2)$ perturbations of ξ_l $(l = 2, 3, \cdots, L-1)$, we find the one that offers the maximum reduction in the objective value and update the corresponding ξ_{l^*} with this new value for future iterations.

Transformation of $\boldsymbol{p}_i(t)$ Similar to the transformation from $\boldsymbol{c}_i(t)$ to $\tilde{\boldsymbol{c}}_i(t)$, we no longer need to maintain each RE's position information in $\boldsymbol{p}_i(t)$ (due to bit interleaving operation). Instead, we only use a single scalar, denoted by $\tilde{p}_i(t)$ (a reduction from the vector notation $\tilde{\boldsymbol{p}}_i(t)$), to indicate the percentage of REs in $e_i(t)$ that is punctured by URLLC (based on $\boldsymbol{p}_i(t)$).

To better understand this transformation, consider an RE that has been punctured by URLLC. For this RE, its effective PD-SNR (dB) is $-\infty$, since on this RE the useful signal power for eMBB transmission is zero and the inference power increases significantly due to URLLC transmission. In this regard, we can imagine $\tilde{p}_i(t)$ corresponds to a "point" $(-\infty)$ on the leftmost position of the $(L+1)$ intervals $((-\infty, \xi_1], \cdots, (\xi_L, +\infty))$ with a probability

$1 - \sum_{k=1}^{L+1} \tilde{c}_{i,k}(t)$.

Based on the above discussions, we have

$$\tilde{p}_i(t) = 1 - \sum_{k=1}^{L+1} \tilde{c}_{i,k}(t). \tag{5.5}$$

Putting Everything Together We have the following new input representation $\tilde{\boldsymbol{s}}_i(t)$:

$$\tilde{\boldsymbol{s}}_i(t) = [\tilde{B}_i(t); \tilde{\boldsymbol{c}}_i(t); \tilde{p}_i(t)], \tag{5.6}$$

where $\tilde{\boldsymbol{c}}_i(t)$ is defined in (5.4), $\tilde{p}_i(t)$ is defined in (5.5), and

$$\tilde{B}_i(t) = \frac{|\mathcal{B}_i(t)|}{|\mathcal{B}|}.$$

We use $\tilde{B}_i(t)$ instead of $|\mathcal{B}_i(t)|$ to ensure all entries in $\tilde{\boldsymbol{s}}_i(t)$ are normalized within $[0, 1]$.

5.5.3 A Case Study

We now use a case study to validate the efficacy of our proposed method to optimize $\boldsymbol{\xi}$ with respect to problem OPT-L. We employ a vector $\boldsymbol{\xi}$ of size $L = 100$ to divide the range $(-\infty, +\infty)$ into 101 intervals: $(-\infty, \xi_1]$, $(\xi_1, \xi_2]$, $\cdots$, $(\xi_{100}, +\infty)$.

As the first step of the method, we collect a number $N = 10^6$ of per-RE PD-SNR samples from 10 eMBB users (see Section 5.9 for parameter settings). During data collection, the 10 eMBB users randomly roam within the cell and the PD-SNRs from all users in each TTI are stored into the sample set $\mathcal{N}$. Then the PD-SNR samples in $\mathcal{N}$ are sorted in ascending order.

After data collection, the elements in $\boldsymbol{\xi}$ are initialized such that all intervals $(-\infty, \xi_1]$, $\cdots$,

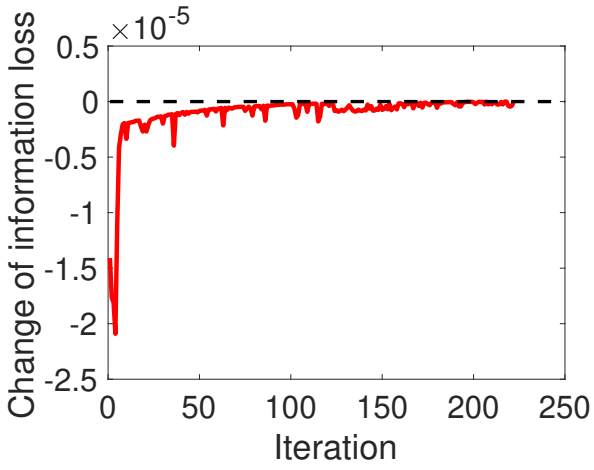


Figure 5.6: The change of information loss through the iterations.

$(\xi_{100}, +\infty)$ contain the same number of PD-SNR samples in $\mathcal{N}$. We then execute the greedy algorithm described in Section 5.5.2 to optimize $\boldsymbol{\xi}$ for a maximum 10^4 iterations. In each iteration, a chosen entry ξ_{l^*} ($l^* \in \{2, \cdots, 99\}$) in $\boldsymbol{\xi}$ that achieves the maximum reduction in the objective value of OPT-L is perturbed by an amount corresponding to 10 PD-SNR samples in the sorted set $\mathcal{N}$.

In Fig. 5.6, we show the change of information loss (i.e., the change of the objective value of OPT-L) achieved by the proposed greedy algorithm in each iteration. We can see that the change of information loss is always negative, meaning that the information loss can be effectively reduced through the iterations. In particular, the algorithm terminates at iteration 223 because the objective of OPT-L cannot be further reduced by adjusting any entry in $\boldsymbol{\xi}$.

The vector $\boldsymbol{\xi}$ optimized in this case study will be used for our performance evaluation of DELUXE in Section 5.9.

5.6 Learning and Prediction

This section presents the details of DFA μ, which is Stage II of DELUXE in Fig. 5.5. Our goal is to obtain an optimized μ that is able to map an input vector $\tilde{\boldsymbol{s}}_i(t)$ into a vector of $(1 - r_{i,m}(t))$ predictions for each MCS $m \in \mathcal{M}$, where $r_{i,m}(t)$ is the BLER of $e_i(t)$ under MCS m (see (5.1)).

Motivation Assume that there exists an *unknown* function, denoted by $f(\cdot)$, that maps each input vector $\tilde{\boldsymbol{s}}_i(t)$ to a vector of *actual* $(1 - r_{i,m}(t))$ values for each MCS $m \in \mathcal{M}$. That is,

$$\boldsymbol{q}_i(t) = f(\tilde{\boldsymbol{s}}_i(t)), \tag{5.7}$$

where $\boldsymbol{q}_i(t)$ is an $M \times 1$ vector, defined as

$$\boldsymbol{q}_i(t) = \begin{bmatrix} 1 - r_{i,1}(t) \\ 1 - r_{i,2}(t) \\ \cdots \\ 1 - r_{i,M}(t) \end{bmatrix} \tag{5.8}$$

If function f is known, one can directly find the optimal MCS based on the BLER target γ in (5.1). However, it is analytically intractable to determine f since $r_{i,m}(t)$ cannot be modeled by a closed-form mathematical expression. Further, due to the large input space of $\tilde{\boldsymbol{s}}_i(t)$, we cannot model f empirically through simulations or experiments.

The above difficulties motivate us to employ deep learning to approximate f with a DFA μ. This approach allows us to learn a good approximation for f using a relatively small set of $\tilde{\boldsymbol{s}}_i(t)$ samples from the input space. After the learning phase, μ will be able to provide good predictions for $(1 - r_{i,m}(t))$ even when the input vectors are outside the sample set. In other words, the DFA μ is capable of achieving statistical generalization (in terms of serving

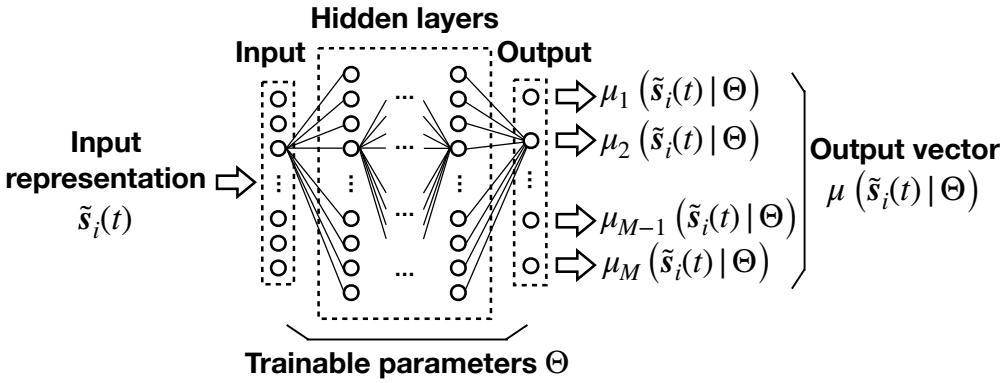


Figure 5.7: Structure of DFA μ.

as a function approximation machine) beyond the sample set used for learning [95].

Key Idea The DFA μ that we use to approximate f is a *feedforward neural network* consisting of an input layer, a number of hidden layers and an output layer, as shown in Fig. 5.7.[5] Within μ's multi-layer structure, there is a set Θ of trainable parameters that determine the mapping from the input to the output of μ. Given an input vector $\tilde{\boldsymbol{s}}_i(t)$, the output of μ is a vector of size M, defined as

$$\mu\left(\tilde{\boldsymbol{s}}_i(t)|\Theta\right) = \begin{bmatrix} \mu_1\left(\tilde{\boldsymbol{s}}_i(t)|\Theta\right) \\ \mu_2\left(\tilde{\boldsymbol{s}}_i(t)|\Theta\right) \\ \cdots \\ \mu_M\left(\tilde{\boldsymbol{s}}_i(t)|\Theta\right) \end{bmatrix} \tag{5.9}$$

where $\mu_m\left(\tilde{\boldsymbol{s}}_i(t)|\Theta\right)$ $(m = 1, \cdots, M)$ denotes the m-th output entry in $\mu\left(\tilde{\boldsymbol{s}}_i(t)|\Theta\right)$.

Ideally, an optimal set of parameters, denoted by Θ^*, should satisfy:

$$\mathbb{E}\Big[\boldsymbol{q}_i(t) - \mu(\tilde{\boldsymbol{s}}_i(t)|\Theta^*)\Big] = \mathbf{0}, \tag{5.10}$$

[5] A key consideration in choosing the multi-layer structure of μ is that the inference time of μ (forward propagation from input layer to output layer) should meet the real-time requirement of 100 μs. The multi-layer structure used in our experiments will be presented in Section 5.9.1.

where the expectation $\mathbb{E}[\cdot]$ is with respect to all possible realizations of $\tilde{\boldsymbol{s}}_i(t)$. In this case, $\mu(\tilde{\boldsymbol{s}}_i(t)|\Theta^*)$ can be used as an approximation for $\boldsymbol{q}_i(t)$ for any $e_i(t)$.

Unfortunately, it is impossible to directly learn Θ^* based on (5.10). This is because for a given input $\tilde{\boldsymbol{s}}_i(t)$, we cannot determine $\boldsymbol{q}_i(t)$ either analytically or empirically.

Nevertheless, (5.10) points us a direction on how to obtain an approximation for Θ^*. First, we establish a relationship between $(1-r_{i,m}(t))$ (i.e., entries in $\boldsymbol{q}_i(t)$) and the decoding result of $e_i(t)$. Denote $a_{i,m}(t)$ as the binary decoding result for $e_i(t)$ under MCS m as follows:

$$a_{i,m}(t) = \begin{cases} 1, & \text{if decoding of } e_i(t) \text{ is successful,} \\ 0, & \text{otherwise.} \end{cases}$$

Then the decoding success probability for $e_i(t)$ under MCS m is equal to the expectation of $a_{i,m}(t)$, i.e., $\mathbb{E}\left[a_{i,m}(t)|\tilde{\boldsymbol{s}}_i(t), m\right]$, where the expectation is conditioned on $\tilde{\boldsymbol{s}}_i(t)$ and m. Since $r_{i,m}(t)$ represents the actual BLER for $e_i(t)$ under MCS m, we have the following relationship between $a_{i,m}(t)$ and $r_{i,m}(t)$:

$$\mathbb{E}\left[a_{i,m}(t)|\tilde{\boldsymbol{s}}_i(t), m\right] = 1 - r_{i,m}(t). \tag{5.11}$$

Then substituting each entry $(1-r_{i,m}(t))$ in $\boldsymbol{q}_i(t)$ with $\mathbb{E}\left[a_{i,m}(t)|\tilde{\boldsymbol{s}}_i(t), m\right]$ in (5.10), we have:

$$\mathbb{E}\Big[\mathbb{E}\left[a_{i,m}(t)|\tilde{\boldsymbol{s}}_i(t), m\right] - \mu_m\left(\tilde{\boldsymbol{s}}_i(t)|\Theta^*\right)\Big|m\Big] = 0, \tag{5.12}$$

for each $m \in \mathcal{M}$. But (5.12) also cannot be used to learn Θ^* since $\mathbb{E}\left[a_{i,m}(t)|\tilde{\boldsymbol{s}}_i(t), m\right]$ is unknown (for the same reason as for $\boldsymbol{q}_i(t)$).

Although $\mathbb{E}\left[a_{i,m}(t)|\tilde{\boldsymbol{s}}_i(t), m\right]$ in (5.12) is unknown, $a_{i,m}(t)$ for each TTI is known. So instead of using $\mathbb{E}\left[a_{i,m}(t)|\tilde{\boldsymbol{s}}_i(t), m\right]$ in (5.12), we propose to use $a_{i,m}(t)$ in the learning method.

Our goal is to learn a set of parameters $\hat{\Theta}^*$ to approximate the optimal Θ^*. Specifically, our learning method will ensure that $\hat{\Theta}^*$ satisfies:

$$\mathbb{E}\Big[a_{i,m}(t) - \mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)\Big] = 0, \tag{5.13}$$

where the expectation is taken with respect to $e_i(t)$, MCS m, and $a_{i,m}(t)$. For (5.13), we have:

$$\begin{aligned} & \mathbb{E}\Big[a_{i,m}(t) - \mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)\Big] \\ = \; & \mathbb{E}\Big[\mathbb{E}\big[a_{i,m}(t)|\tilde{\boldsymbol{s}}_i(t), m\big] - \mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)\Big] \\ = \; & \mathbb{E}\Big[1 - r_{i,m}(t) - \mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)\Big] \\ = \; & 0\,. \end{aligned} \tag{5.14}$$

Comparing (5.12) and (5.14), we can see that Θ^* is more accurate than $\hat{\Theta}^*$ in terms of predicting $(1 - r_{i,m}(t))$. This is because Θ^* guarantees the prediction accuracy for *every* MCS $m \in \mathcal{M}$. In contrast, $\hat{\Theta}^*$ only ensures that the average gap between $(1 - r_{i,m}(t))$ and $\mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)$ over *all* MCSs is zero. But as Θ^* cannot be obtained in practice, we will find $\hat{\Theta}^*$ instead and use it as an approximation for Θ^*.

In the rest of this section, we will present the details of our learning method.

Some Details Since the learning for DFA μ is best performed through gradient-descent approaches [95], we propose to learn parameters $\hat{\Theta}^*$ by solving the following mean-squared error (MSE) minimization problem instead of the equation in (5.13):

$$\text{minimize} \quad G(\Theta) = \mathbb{E}\left[\left(a_{i,m}(t) - \mu_m\left(\tilde{\boldsymbol{s}}_i(t)|\Theta\right)\right)^2\right], \tag{5.15}$$

Algorithm 5

% Data Collection:
Empty replay buffer;
for $t = 1, 2, \cdots, T$ **do**
    Receive eMBB state representation $\tilde{\boldsymbol{s}}_t$;
    Select an MCS m_t from $\mathcal{M}$ following a uniform distribution;
    Transmit eMBB and receive its decoding result a_t via feedback ;
    Store sample tuple $(\tilde{\boldsymbol{s}}_t, m_t, a_t)$ in the replay buffer;
end for
% Learning Phase:
Initialize Θ_0 with random parameters;
for $j = 1, 2, \cdots, J$ **do**
    Randomly choose a mini-batch of K samples from the replay buffer;
    Update the parameters Θ_j using $\mathrm{SG}(\Theta_{j-1})$ following (5.18);
end for
return $\hat{\Theta}^* = \Theta_J$.

where the expectation is with respect to all possible realizations of $e_i(t)$, MCS m, and $a_{i,m}(t)$.

To see why parameters $\hat{\Theta}^*$ as characterized in (5.13) can be obtained through solving the MSE minimization problem (5.15), consider the gradient of $G(\Theta)$ with respect to Θ:

$$\begin{aligned} \nabla_\Theta G(\Theta) &= \mathbb{E}\Big[-2\big(a_{i,m}(t) - \mu_m\left(\tilde{\boldsymbol{s}}_i(t)|\Theta\right)\big) \cdot \\ &\qquad \nabla_\Theta \mu_m\left(\tilde{\boldsymbol{s}}_i(t)|\Theta\right)\Big]. \end{aligned} \tag{5.16}$$

Assume that $\hat{\Theta}^*$ is the solution to (5.15). Then it must satisfy $\nabla_\Theta G(\Theta)|_{\Theta=\hat{\Theta}^*} = \mathbf{0}$. In (5.16), $\nabla_\Theta \mu_m\left(\tilde{\boldsymbol{s}}_i(t)|\Theta\right)$ is a vector of the gradient of $\mu_m\left(\tilde{\boldsymbol{s}}_i(t)|\Theta\right)$ with respect to the parameters in θ and cannot be forced to be zero. Thus we must have

$$\mathbb{E}\Big[a_{i,m}(t) - \mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)\Big] = 0,$$

which is exactly (5.13).

We now propose a gradient-descent method to learn $\hat{\Theta}^*$. This method consists of a *data*

collection phase and a *learning* phase. The data collection phase is to gather a large set of eMBB transmission samples from the cell and store them into a replay buffer. The learning phase is to learn $\hat{\Theta}^*$ through iterative gradient descent based on the collected eMBB samples.

The data collection phase involves a total of T TTIs (e.g., $T = 10^5 \sim 10^6$). In each TTI $t = 1, \cdots, T$, an eMBB transmission sample $(\tilde{\boldsymbol{s}}_t, m_t, a_t)$ (notation is reduced for the ease of exposition) is obtained and stored into the replay buffer. In particular, the MCS m_t for each eMBB transmission sample is chosen from $\mathcal{M}$ following a uniform distribution. Such a uniformly random MCS selection is to ensure that all MCSs in $\mathcal{M}$ can be thoroughly explored during data collection.

After data collection is completed, we start the learning phase to determine $\hat{\Theta}^*$. The learning phase involves a total of J iterations, where J should be sufficiently large (e.g., $J = 10^5 \sim 10^6$) to achieve good performance. Denote Θ_j as the set of parameters obtained in the j-th iteration. Before the first iteration, we initialize parameters Θ_0 randomly. For the j-th iteration ($j = 1, 2, \cdots, J$), we update parameters Θ_j by taking a small step along the gradient $\nabla_\Theta G(\Theta)|_{\Theta=\Theta_{j-1}}$ given in (5.16).

Since it is intractable to determine the expectation in $\nabla_\Theta G(\Theta)|_{\Theta=\Theta_{j-1}}$, we perform the *stochastic gradient descent (SGD)* [95] for each update of Θ_j. Specifically, in each iteration, we randomly choose a set of K (e.g., $K = 32$) samples $(\tilde{\boldsymbol{s}}_k, m_k, a_k)$, $k = 1, \cdots, K$ from the replay buffer to approximate $\nabla_\Theta G(\Theta)|_{\Theta=\Theta_{j-1}}$ using the following sample gradient $\mathrm{SG}(\Theta_{j-1})$:[6]

$$\begin{aligned} \mathrm{SG}(\Theta_{j-1}) \;=\; & \frac{1}{K}\sum_{k=1}^{K}\Big[-2\big(a_k - \mu_{m_k}(\tilde{\boldsymbol{s}}_k|\Theta_{j-1})\big)\cdot \\ & \nabla_\Theta \mu_{m_k}(\tilde{\boldsymbol{s}}_k|\Theta)\,\big|_{\Theta=\Theta_{j-1}}\Big]. \end{aligned} \tag{5.17}$$

[6]The purpose of random sampling is to break the temporal correlations among the samples and improve the stability of learning [85, 116].

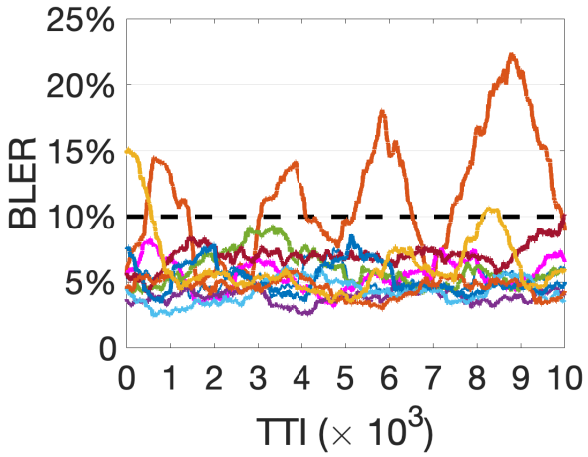


Figure 5.8: BLER performance of 10 eMBB users without calibrating μ's approximation error.

Then we update Θ_j as follows:

$$\Theta_j = \Theta_{j-1} + \rho \cdot \mathrm{SG}(\Theta_{j-1}), \tag{5.18}$$

where ρ denotes the step size (or learning rate), e.g., $\rho = 10^{-3}$. After the J learning iterations, we set $\hat{\Theta}^* = \Theta_J$. A pseudo code for the data collection and learning phases is given in Algorithm 5.

5.7 Calibration and MCS Selection

Motivation For each eMBB transmission $e_i(t)$, the DFA μ learned in Section 5.6 provides us an approximation $\mu(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)$ for the $(1 - r_{i,m}(t))$ values in $\boldsymbol{q}_i(t)$ (see (5.8)). However, due to the nature of DFA, there likely exists approximation error in $\mu(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)$. As a result, if we directly use $(1 - \mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*))$ to substitute $r_{i,m}(t)$ in MCS selection (based on (5.1)), the actual BLER performance of the eMBB link may not meet the target γ.

As an example, Fig. 5.8 shows the BLER curves of 10 eMBB users (see Section 5.9

for parameter settings)[7], where the MCS is selected for each $e_i(t)$ based directly on $(1 - \mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*))$ $(m = 1, \cdots, M)$ and a BLER target $\gamma = 10\%$. As we see in the figure, there are two users' BLER curves (orange and yellow) that violate the target γ during certain periods. This is because during these periods, μ underestimates their $r_{i,m}(t)$ for each MCS and as result, MCS selection based on such $(1-\mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*))$'s cannot meet the 10% BLER target.

The results in Fig. 5.8 demonstrate that it is necessary to include a calibration mechanism to address the approximation error of μ. In the rest of this section, we describe our proposed calibration mechanism.

Key Idea For a given $e_i(t)$, the approximation error in each of μ's output entry $\mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)$ is:

$$\alpha_m(\tilde{\boldsymbol{s}}_i(t)) = 1 - r_{i,m}(t) - \mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*).$$

But it is impossible to calculate the approximation error $\alpha_m(\tilde{\boldsymbol{s}}_i(t))$ because we do not know $r_{i,m}(t)$.

Since we cannot find $\alpha_m(\tilde{\boldsymbol{s}}_i(t))$ for a given $\tilde{\boldsymbol{s}}_i(t)$, one may try to find an average approximation error over all possible $\tilde{\boldsymbol{s}}_i(t)$. Denote this average approximation error as α_m. Then we have:

$$\begin{aligned}
\alpha_m &= \mathbb{E}\Big[1 - r_{i,m}(t) - \mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)\Big|m\Big] \\
&= \mathbb{E}\Big[\mathbb{E}\left[a_{i,m}(t)|\tilde{\boldsymbol{s}}_i(t), m\right] - \mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)\Big|m\Big] \\
&= \mathbb{E}\Big[a_{i,m}(t) - \mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)\Big|m\Big]
\end{aligned}$$

where the second equality follows (5.11), and the third equality follows the law of total

[7]The BLER curves in Fig. 5.8 are computed using a sliding window with a size of 1,000 TTIs.

expectation.

Unfortunately, it is not feasible to estimate and calibrate such per-MCS (m) based approximation errors. This is because there is simply not enough decoding results ($a_{i,m}(t)$'s) available for each MCS $m \in \mathcal{M}$ in the desired time scale of our interest. For example, over certain period, there may be no user present in the cell that has strong enough channel quality to support higher MCS levels (e.g., MCS 17-28 with 64-QAM modulation (Table 5.1.3.1-1 in [36]). As a result, we cannot accurately estimate α_m's in a timely fashion.

Although it is infeasible to estimate approximation error for each m, it is possible (and practical) to develop a per-user based approach to calibrate the average approximation error. Specifically, we introduce a calibration parameter β^x for each eMBB user x. Recall that $\upsilon(e_i(t))$ is the receiving eMBB user corresponding to $e_i(t)$ (see Section 5.3). Then we use β^x to calibrate all μ's output entries $\mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)$ $(m = 1, 2, \cdots, M)$, if $x = \upsilon(e_i(t))$. The goal is to set β^x properly so that the following mean squared approximation error is minimized for each user x:

$$y^x = \mathbb{E}\Big[\mathbb{E}\Big[\Big(1 - r_{i,m}(t) - \big(\mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*) + \beta^x\big)\Big)^2 \Big| m\Big]\Big] \,. \tag{5.19}$$

In (5.19), the inner expectation (conditioned on a specific MCS m) is with respect to all possible realizations of $\tilde{\boldsymbol{s}}_i(t)$ for user x, and the outer expectation is with respect to all MCS m.

With this calibration, the calibrated BLER prediction $\tilde{r}_{i,m}(t)$ for a given $e_i(t)$ and MCS $m \in \mathcal{M}$ is:

$$\tilde{r}_{i,m}(t) = 1 - \Big(\mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*) + \beta^x\Big) \,, \tag{5.20}$$

where $x = \upsilon(e_i(t))$. Then the MCS selection strategy in (5.1) becomes

$$\hat{m}_i^*(t) = \arg\max_{m \in \mathcal{M}} \text{SE}(m), \text{ subject to : } \tilde{r}_{i,m}(t) \leq \gamma, \tag{5.21}$$

where $\hat{m}_i^*(t)$ denotes the MCS level selected for $e_i(t)$.

Optimize β^x The optimization of β^x with the objective of minimizing y^x in (5.19) can be performed through an iterative gradient descent approach. Specifically, the gradient of y^x with respect to β^x can be obtained as

$$\begin{aligned}
\frac{dy^x}{d\beta^x} &= \mathbb{E}\Big[\mathbb{E}\Big[-2\Big(1 - r_{i,m}(t) - \\
&\quad \big(\mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*) + \beta^x\big)\Big)\Big|m\Big]\Big] \\
&= \mathbb{E}\Big[\mathbb{E}\Big[-2\Big(\mathbb{E}\left[a_{i,m}(t)|\tilde{\boldsymbol{s}}_i(t), m\right] - \\
&\quad \big(\mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*) + \beta^x\big)\Big)\Big|m\Big]\Big] \\
&= \mathbb{E}\Big[\mathbb{E}\Big[-2\Big(a_{i,m}(t) - \big(\mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*) + \beta^x\big)\Big)\Big|m\Big]\Big],
\end{aligned} \tag{5.22}$$

where the second equality follows (5.11), and the third equality follows the law of total expectation. We use the sign function $\text{sgn}(\cdot)$ to indicate whether $\frac{dy^x}{d\beta^x}$ is positive or negative:

$$\text{sgn}(\frac{dy^x}{d\beta^x}) = \begin{cases} 1, & \text{if } \dfrac{dy^x}{d\beta^x} > 0, \\ 0, & \text{if } \dfrac{dy^x}{d\beta^x} = 0, \\ -1, & \text{otherwise.} \end{cases}$$

Then in each iteration, we update β^x as follows:

$$\beta^x \leftarrow \beta^x - \text{sgn}(\frac{dy^x}{d\beta^x}) \cdot \Delta, \tag{5.23}$$

where Δ denotes the step size.

We now show how to incorporate this iterative gradient descent into our 5G setting. Specifically, We show how to implement (5.23) as a simple additive increase/decrease (AID) mechanism to periodically update β^x for each eMBB user $x \in \mathcal{X}$.

Since the gradient $\frac{dy^x}{d\beta^x}$ cannot be determined analytically, we perform SGD using the recent transmission results of each user x to update β^x periodically. Specifically, at the BS, we maintain a buffer $\mathcal{Q}^x$ for each user x. When the BS receives a decoding result $a_{i,m}(t)$ for a transmission $e_i(t)$, the scheduler pushes the result into the corresponding user $\upsilon(e_i(t))$'s buffer $\mathcal{Q}_{\upsilon(e_i(t))}$. When a new decoding result arrives at the buffer and the buffer is full, the the oldest decoding result in the buffer will be pushed out.

For each user x, β^x is updated after every Z transmissions (e.g., $Z = 10$), which correspond to Z decoding results in $\mathcal{Q}^x$. Whenever there are Z new results in $\mathcal{Q}^x$, denoted by a_z^x, $z = 1 \cdots, Z$, we compute the sample gradient as follows:

$$\begin{aligned} \text{SY}^x &= \frac{1}{Z}\sum_{z=1}^{Z}[-2\,(a_z^x - (1-\gamma))], \\ &= 2(1-\gamma-\frac{1}{Z}\sum_{z=1}^{Z} a_z^x) \\ &= 2(1-\gamma-A^x(Z)), \end{aligned}$$

where $(1-\gamma)$ is used to approximate $\left(\mu_m(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*) + \beta^x\right)\Big)$ in (5.22) for each of the Z transmissions, and $A^x(Z)$ denotes $\sum_{z=1}^{Z} a_z^x/Z$, i.e., the mean of the most recent Z decoding

Algorithm 6 Calibration and MCS selection

Initialization: For each eMBB user x, set $d^x = 0$, $\beta^x = 0$;
%MCS selection
upon event an $e_i(t)$ is scheduled for transmission **do**
    $x = \upsilon(e_i(t))$;
    Obtain input representation $\tilde{\boldsymbol{s}}_i(t)$ as (5.6);
    Obtain the prediction $\mu(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)$;
    Obtain the calibrated BLER $\tilde{r}_{i,m}(t)$ based on β^x and (5.20)
    Select MCS $\hat{m}_i^*(t)$ based on (5.21);
    Transmit $e_i(t)$ using MCS $\hat{m}_i^*(t)$;
%β^x adjustment
upon event a decoding result $a_{i,m}(t)$ is received **do**
    $x = \upsilon(e_i(t))$;
    Push $a_{i,m}(t)$ into the buffer $\mathcal{Q}^x$;
    Counter $d^x \leftarrow d^x + 1$;
    if $(d^x \bmod N) = 0$ **then**
        Update β^x following (5.25);
    end if

results in $\mathcal{Q}^x$. The use of $(1-\gamma)$ as an approximation follows from (5.20) and (5.21), where an MCS m is chosen corresponding to the largest $\tilde{r}_{i,m}(t)$ that is less than γ. Then β^x is updated as follows:

$$\beta^x \leftarrow \beta^x - \text{sgn}(1 - A^x(Z) - \gamma) \cdot \Delta \ . \tag{5.24}$$

In (5.24), $(1 - A^x(Z))$ is the empirical decoding failure rate among the Z transmissions.

However, if (5.24) is directly used to update β^x, β^x will experience frequent fluctuations, bringing BLER of each user frequently exceeding target γ. This is because the sgn function is extremely sensitive: when the decoding failure rate is slightly lower than γ, which is perfectly fine in practice, β^x will still be forced to increase. As a result, the decoding failure rate of the subsequent transmissions will increase immediately, leading to frequent violations of target γ.

To address this undesirable sensitivity in (5.24), we introduce a parameter γ_0 as a cushion when assessing current decoding failure rate and adjusting β^x. Instead of adjusting β^x when

the decoding failure rate falls below γ, we will wait and make adjustment only when the decoding failure rate falls below $(\gamma - \gamma_0)$.

Furthermore, we propose to use two different step sizes, Δ_{D} and Δ_{I}, for the decreasse and increase of β^x during adjustment, respectively. In particular, to ensure a fast response in the event that a user's decoding failure rate goes beyond the target γ, we set $\Delta_{\text{D}} \geq \Delta_{\text{I}}$.

Based on the above discussion, we have the following improved adjustment algorithm for β^x:

$$\beta^x \leftarrow \begin{cases} \beta^x - \Delta_{\text{D}}, & \text{if } 1 - A^x(Z) > \gamma, \\ \beta^x + \Delta_{\text{I}}, & \text{if } 1 - A^x(Z) < \gamma - \gamma_0. \end{cases} \tag{5.25}$$

Summary A pseudo code summarizing the MCS selection and β^x adjustment for each eMBB user $x \in \mathcal{X}$ is given in Algorithm 6.

5.8 Real-time Implementation

In Sections 5.5 to 5.7, we described in detail the three stages of DELUXE to select an MCS for an eMBB transmission $e_i(t)$. In each TTI, there are usually multiple eMBB transmissions scheduled, i.e., $|\mathcal{E}(t)| > 1$. Given that the shortest duration of a TTI for eMBB is 125 μs (under NR numerology 3), the implementation problem we need to address is to ensure the MCS selection for all $e_i(t) \in \mathcal{E}(t)$ scheduled for a TTI t can be completed within 100 μs time scale.

To address this problem, we propose a hybrid CPU and GPU-based implementation as follows:

Step (i) The input representation vectors $\tilde{\boldsymbol{s}}_i(t)$ for all $e_i(t) \in \mathcal{E}(t)$ are generated by the

CPU and then transferred to the GPU.

Step (ii) We use a GPU to compute $\mu(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)$ for all $e_i(t) \in \mathcal{E}(t)$ concurrently (more details to follow). The motivation of employing a GPU for this step is to exploit parallelism to reduce the computation time. Specifically, the computations of $\mu(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)$ for all $e_i(t) \in \mathcal{E}(t)$ can be done in parallel through matrix operations. This can be handled much more efficiently by a GPU than a CPU.

Step (iii) Based on the computed $\mu(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)$ values, MCS selections for all $e_i(t) \in \mathcal{E}(t)$ are performed in parallel on the GPU following (5.21). Then the result of these selected MCSs is transferred back to the CPU. The update of β^x for each eMBB user x is performed on the CPU and the β^x values are sent to the GPU after each update.

We now describe Step (ii) in detail, i.e., how to compute $\mu(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)$ for all $e_i(t) \in \mathcal{E}(t)$ concurrently using parallel threads in a GPU. Recall that μ is a fully-connected neural network consisting of an input layer, a number of hidden layers, and an output layer. Denote $\boldsymbol{W}_{(j,j+1)}$ as the weight matrix for the connections from the neurons in the j-th layer to the neurons in the $(j+1)$-th layer. For instance, $\boldsymbol{W}_{(1,2)}$ is the weight matrix for the connections from the input layer to the first hidden layer. The element in the k-th row and the l-th column of the matrix $\boldsymbol{W}_{(j,j+1)}$ is the weight on the connection from the k-th neuron in the j-th layer to the l-th neuron in the $(j+1)$-th layer. Further, denote $\boldsymbol{\pi}_j$ as a vector of the bias added to each neuron in the j-th layer, and $\phi(\cdot)$ as the non-linear activation function (e.g., rectified non-linearity) used for the hidden layers.

Consider a specific $e_i(t) \in \mathcal{E}(t)$. The first step to compute $\mu(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)$ for $e_i(t)$ is to feed the vector $\tilde{\boldsymbol{s}}_i(t)$ into the input layer of μ. Then the forward propagation from the input layer to the first hidden layer is computed as: $\phi(\boldsymbol{W}_{(1,2)} \cdot \tilde{\boldsymbol{s}}_i(t) + \boldsymbol{\pi}_2)$. Subsequently, the forward propagation to the second hidden layer is computed as: $\phi(\boldsymbol{W}_{(2,3)} \cdot \phi(\boldsymbol{W}_{(1,2)} \cdot \tilde{\boldsymbol{s}}_i(t) + \boldsymbol{\pi}_2) + \boldsymbol{\pi}_3))$.

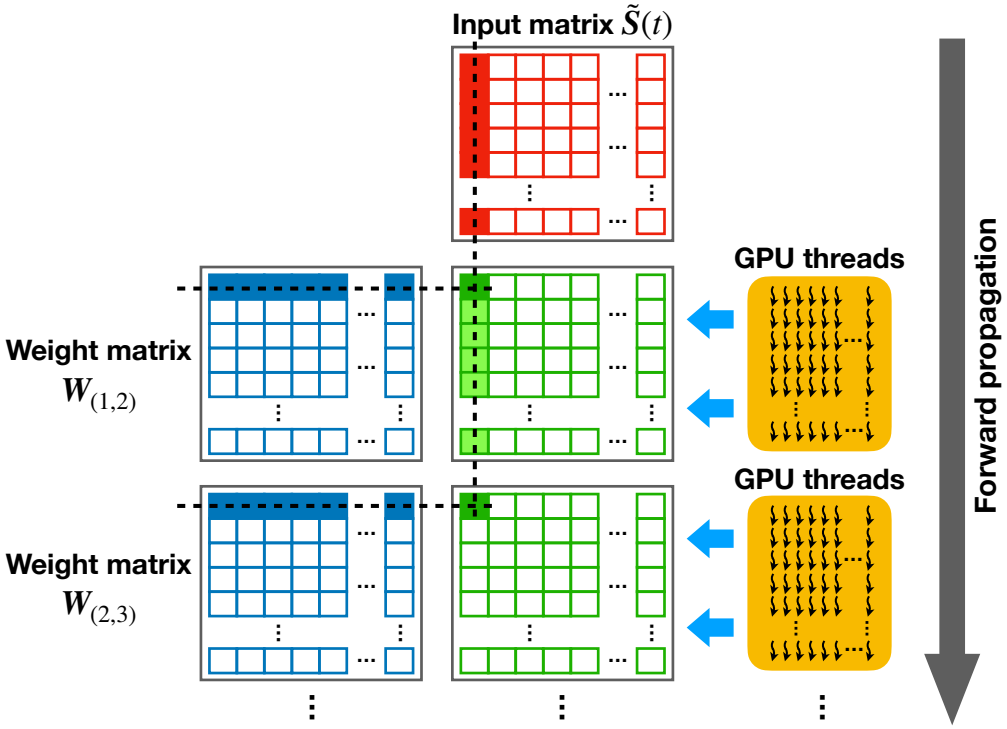


Figure 5.9: Concurrent forward propagation for all $e_i(t) \in \mathcal{E}(t)$.

Such computations proceed through the remaining layers in μ until we obtain the final output $\mu(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)$ for $e_i(t)$.

The above forward propagation process for all $e_i(t) \in \mathcal{E}(t)$ can be implemented concurrently through matrix operations leveraging GPU parallel processing. An illustration for this concurrent forward propagation is given in Fig. 5.9. Specifically, we combine the vectors $\tilde{\boldsymbol{s}}_i(t)$ for all $e_i(t) \in \mathcal{E}(t)$ into a matrix: $\tilde{\boldsymbol{S}}(t) = [\tilde{\boldsymbol{s}}_1(t); \tilde{\boldsymbol{s}}_2(t); \cdots]$, whose number of columns is equal to $|\mathcal{E}(t)|$. Then the forward propagation from the input layer to the first hidden layer for all $e_i(t) \in \mathcal{E}(t)$ can be computed as: $\phi(\boldsymbol{W}_{(1,2)} \cdot \tilde{\boldsymbol{S}}(t) + \boldsymbol{\Pi}_2)$, where $\boldsymbol{\Pi}_2$ is a matrix defined as $\boldsymbol{\Pi}_2 = [\boldsymbol{\pi}_2; \boldsymbol{\pi}_2; \cdots]$. We employ a two-dimensional array of GPU threads to complete this computation, as shown in Fig. 5.9. Each thread first computes the inner product of a row in $\boldsymbol{W}_{(1,2)}$ and a column in $\tilde{\boldsymbol{S}}(t)$, and then adds the product (a scalar) with the corresponding bias element in $\boldsymbol{\Pi}_2$. Finally, the non-linear activation $\phi(\cdot)$ is applied to the result.

After all threads finish computating $\phi(\boldsymbol{W}_{(1,2)} \cdot \tilde{\boldsymbol{S}}(t) + \boldsymbol{\Pi}_2)$ (a matrix), the forward propagation to the second hidden layer is computed similarly as: $\phi(\boldsymbol{W}_{(2,3)} \cdot \phi(\boldsymbol{W}_{(1,2)} \cdot \tilde{\boldsymbol{S}}(t) + \boldsymbol{\Pi}_2) + \boldsymbol{\Pi}_3)$. Again, we use an array of GPU threads to do this computation. Following the same token, the forward propagation through the subsequent layers in μ proceeds until we obtain an output matrix $[\mu(\tilde{\boldsymbol{s}}_1(t)|\hat{\Theta}^*); \mu(\tilde{\boldsymbol{s}}_2(t)|\hat{\Theta}^*); \cdots]$, whose columns are the vectors $\mu(\tilde{\boldsymbol{s}}_i(t)|\hat{\Theta}^*)$ for all $e_i(t) \in \mathcal{E}(t)$.

Based on this implementation, the execution time of DELUXE only depends on the size of input data L and the number of eMBB transmissions $|\mathcal{E}(t)|$ in a TTI t, which correspond to the numbers of rows and columns in the matrix $\tilde{\boldsymbol{S}}(t)$.

5.9 Performance Evaluation

In this section, we conduct an experimental study to evaluate the performance of DELUXE. We organize this section as follows. In Section 5.9.1, we describe our experimental platform and system settings. In Section 5.9.2, we use a case study to validate the efficacy of DELUXE in solving the MCS selection problem in (5.1). In Section 5.9.3, we further evaluate the performance of DELUXE under varying parameters to further understand its behavior.

5.9.1 Experimental Platform and Settings

Platform All experiments are completed on an NVIDIA DGX station with an Intel Xeon E5-2698 v4 CPU and an NVIDIA Tesla V100 GPU. Programming on GPU is based on NVIDIA CUDA v10.2 [25] with C++. Data communication between CPU and GPU is through a PCIe 3.0 X16 slot with default configuration.

For all experiments, we built a link-level 5G NR simulator that supports dynamic eM-

Table 5.3: Key network settings.

Carrier frequency	4 GHz
NR numerology	Numerology 1 (30 kHz SC spacing)
System bandwidth	20 MHz, 50 RBs in total mBWP: 25 RBs in high frequency range eBWP: 25 RBs in low frequency range
Cell radius	500 meters
eMBB settings	1 TTI = 1 time slot (14 OFDM symbols) BLER target $\gamma = 10\%$ 29 MCS levels from Table 5.1.3.1-1, [36] Full buffer traffic model
URLLC settings	1 mini-slot = 2 OFDM symbols QPSK modulation with a 1/3 LDPC code rate Poisson arrivals of 50-byte packets Each packet transmitted using 25 RBs in mBWP
Fading channel	CDL-C/TDL-C with 300 ns RMS delay spread [40]
Path-loss model	3D UMa NLOS [40]
Channel estimation	Ideal channel estimation
Antenna config.	4 Tx antennas at the BS 2 Rx antennas at each user
BS Tx power	46 dBm
Noise floor	-91.9 dBm

BB/URLLC multiplexing based on URLLC puncturing. The PHY layer signal processing functions and the multi-path channel fading models (CDL and TDL) are provided by MathWorks 5G Toolbox [105].

Network Settings We consider an NR cell consisting of a BS and a varying number of eMBB users that are randomly located within the cell's coverage. Table 5.3 list key parameter settings for the NR cell. Since the carrier frequency is 4 GHz, we employ NR numerology 1 (30 kHz SC spacing) for the channel [32]. Thus, for 20 MHz channel bandwidth, we have 50 RBs. To decouple the impact of a scheduler algorithm on the performance of DELUXE, we

assume in each TTI t, an RB is randomly allocated to the eMBB users following a uniform distribution. An RB can only be allocated to one eMBB user while an eMBB user may be allocated with multiple RBs. The target BLER for eMBB is set to $\gamma = 10\%$.

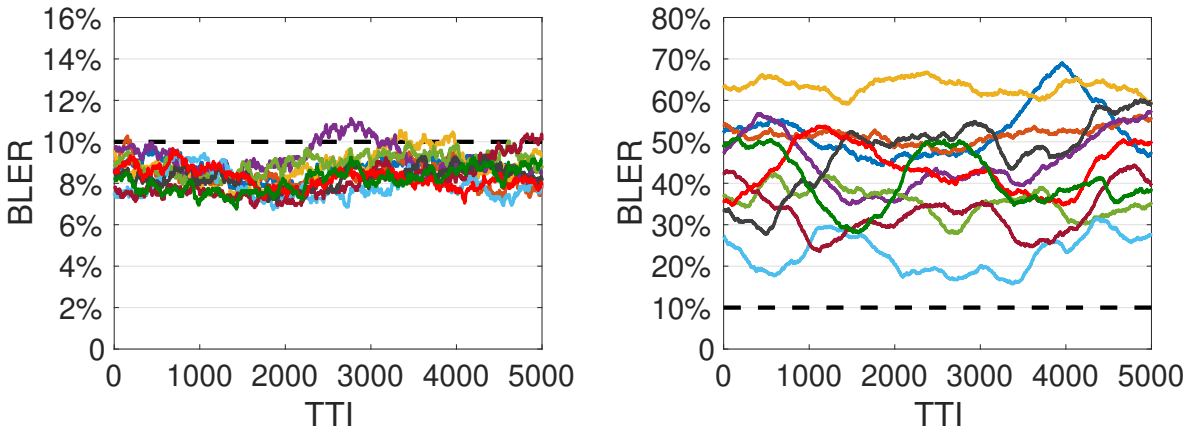
We assume URLLC traffic in the cell follows a Poisson arrival process with a rate of λ (packets per TTI). Following NR standards [2], we set 2 OFDM symbols per mini-slot, a packet size of 50 bytes, QPSK modulation and a 1/3 LDPC code rate for URLLC transmission. Under this setting, each URLLC packet is transmitted using 25 RBs in a mini-slot. Accordingly, we set mBWP (see Fig. 5.1) to 25 RBs.

For multi-path channel fading, we consider two widely used channel models: the *tapped delay line* (TDL) model and the *clustered delay line* (CDL) model [40]. TDL model contains power, delay and Doppler spectrum information for the multi-path channel taps. CDL is a spatial extension of TDL that involves more detailed information such as the departure and arrival angles for each cluster of channel taps. We choose CDL-C and TDL-C channel profiles that are constructed for non-line-of-sight (NLOS) channel conditions.

DELUXE Settings We now describe the setting for each of the three stages of DELUXE (see Fig. 5.5) as follows.

For Stage I, we employ the vector $\boldsymbol{\xi}$ that is optimized in Section 5.5.3.

For Stage II, the DFA μ has two fully-connected hidden layers between its input and output layers, which contain 256 and 128 neurons, respectively. We use rectified non-linearity as the activation function for the neurons in the two hidden layers. To learn parameters $\hat{\Theta}^*$ for μ (refer to Algorithm 5), we use $T = 8 \times 10^5$ TTIs to collect data from 10 eMBB users that randomly roam within the cell. Random RB allocation and random MCS selection are employed for scheduling transmissions to the 10 users. Specifically, we collect data for 2×10^5 TTIs under URLLC arrival rate $\lambda = 1$, 2, 3 and 4, respectively. The replay buffer



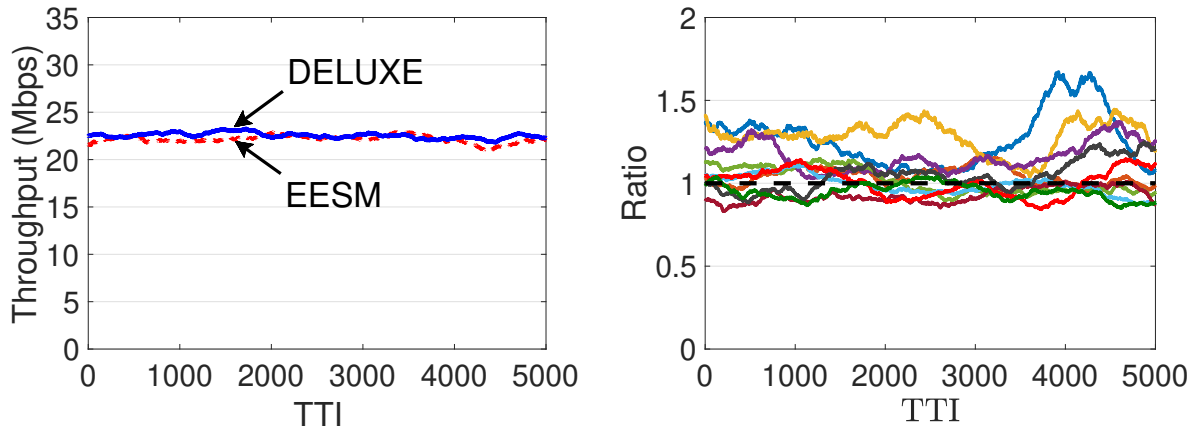
(a) DELUXE (b) EESM

Figure 5.10: BLER performance under DELUXE (a) and EESM (b).

size is set to 8×10^5, which is sufficient to store all collected data. The learning phase involves $J = 4 \times 10^6$ iterations. We use the Adam algorithm [107] with a learning rate of $\rho = 2.5 \times 10^{-4}$ for learning. We set a mini-batch to $K = 64$ samples for each learning iteration.

For Stage III, we set $Z = 10$, $\Delta_{\text{I}} = 2\%$, $\Delta_{\text{D}} = 2.5\%$ and $\gamma_0 = 3\%$.

Benchmarking For performance comparison, we also implemented the EESM method as described in [109, 125] in our 5G simulator. EESM is a popular CQI-based LA method for wideband OFDMA systems such as 4G LTE and 5G NR. The basic idea of EESM is to map a vector of sub-carrier SNRs that are experienced by a codeword into a single *effective SNR* value through exponential functions. Then this effective SNR is used to find the best MCS to meet a given BLER target γ based on the BLER performance curve under each MCS that are measured under AWGN channels. In the absence of URLLC puncturing, EESM can effectively maintain link reliability.



(a) Sum of throughput of 10 eMBB users achieved by DELUXE and EESM.

(b) Normalized throughput of each user under DELUXE with respect to that under EESM.

Figure 5.11: Throughput comparison of DELUXE and EESM.

5.9.2 A Case Study

We now use a case study to evaluate the performance of DELUXE. In this study, we consider $|\mathcal{X}| = 10$ eMBB users in the cell and a URLLC traffic load $\lambda = 2$ packets/TTI. CDL-C channel model is employed in the simulations. In each TTI, DELUXE and EESM experience the same channel conditions, RB allocation to eMBB users, and URLLC packet arrivals.

In Fig. 5.10(a) and (b), we show the BLER performance for the 10 eMBB users over $5,000$ TTIs under DELUXE and EESM, respectively. In both figures, BLER values are calculated using a sliding window with a size of $1,000$ TTIs. In Fig. 5.10(a), we see that DELUXE is able to keep BLER for each user under $\gamma = 10\%$ most of the time. Although several users' BLERs occasionally exceed 10%, they quickly fall back under $\gamma = 10\%$, thanks to the adaptive β^x-based calibration to the MCS selection (Stage III of DELUXE). In contrast, Fig. 5.10(b) shows that under EESM, all users' BLERs exceed (violate) the 10% target threshold throughout 5,000 TTIs. The main reason in EESM's failure is that the effective SNR metric used in EESM cannot capture the impact of URLLC puncturing on each eMBB

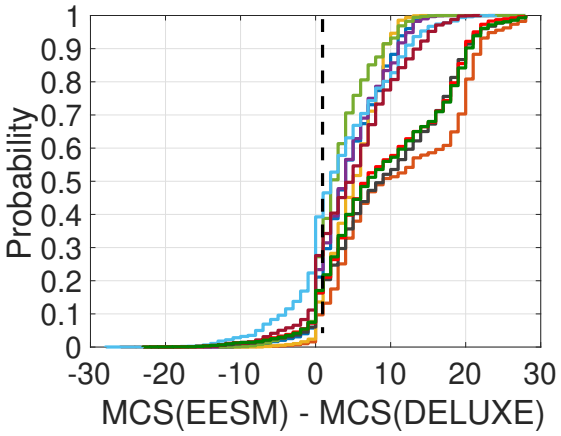


Figure 5.12: CDF of the MCS difference between EESM and DELUXE for each of the 10 users.

transmission.

In Fig. 5.11(a) and (b), we show the throughput performance of the 10 eMBB users under DELUXE and compare it to EESM. Note that throughput here only refers to throughput that is *successfully* decoded at a receiver. Fig. 5.11(a) shows the sum of throughput of the 10 eMBB users achieved by the two schemes. Clearly, they is little difference between the two. Further, in Fig. 5.11(b), we examine the throughput of each individual eMBB user's throughput under DELUXE by normalizing it with respect to that under EESM. Again, the normalized throughput for each user is close to 1, indicating that throughput at per user level is also comparable under the two schemes. These results confirm that while meeting the target $\gamma = 10\%$ BLER threshold, DELUXE can still achieve comparable eMBB throughput as that under EESM, which is not constrained by the BLER constraints. Note that EESM is the state-of-the-art LA method used in 4G LTE and 5G NR and has been well regarded as the most effective method to maintain link reliability while maximizing spectral efficiency (in the absence of URLLC puncturing) [109, 125].

To see how different MCS selection is under the two schemes, we plot the cumulative

distribution function (CDF) of MCS under EESM minus MCS under DELUXE for the 10 users in Fig. 5.12. In this figure, when MCS difference is positive (negative), it means that MCS under EESM is higher (lower) than that under DELUXE. We can see that for each of the 10 users, EESM has a higher probability to select a MCS level greater than DELUXE. This corroborates with the results in Fig. 5.10(b) and (a), which shows that BLER under EESM is much higher than that under DELUXE.

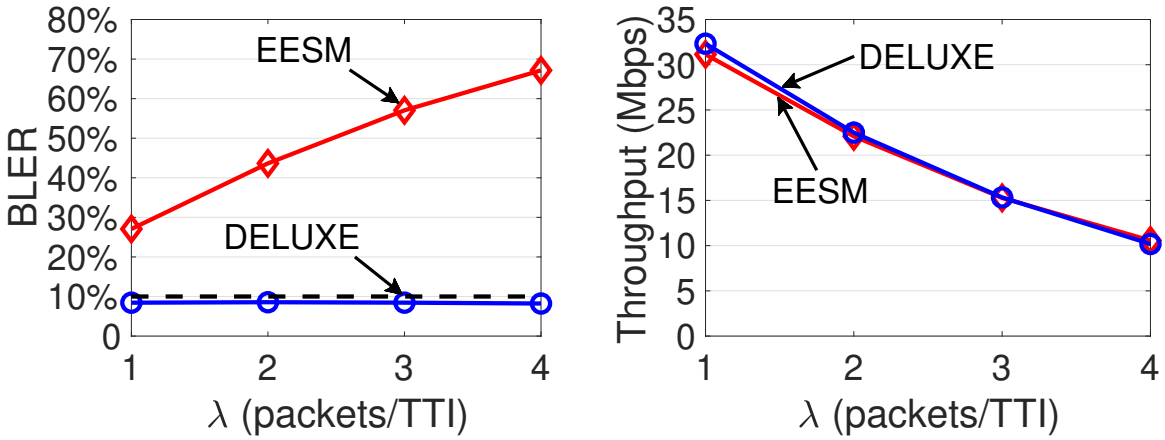
Finally, the total execution time of DELUXE for 10 eMBB is 90.6 μs, which meets the 5G real-time requirement ($\sim$100 μs for numerology 3). The total execution time of DELUXE includes the computation time at both CPU and GPU, and the time cost for data transferring between CPU and GPU.

5.9.3 Varying Network Parameters

In this section, we examine the behavior of DELUXE by varying different parameters such as URLLC traffic arrival rate λ, multi-path channel models, and the number of eMBB users in the cell.

Varying URLLC Traffic Load We vary URLLC traffic arrival rate λ for the same 10 eMBB users in the cell. We assume CDL-C multi-path channel fading model. In Fig. 5.13(a), we plot the average BLER among the 10 users over $5,000$ TTIs achieved by our DELUXE algorithm. We can see that the average BLER under DELUXE always stays below the $\gamma = 10\%$ target when λ increases. Also shown in Fig. 5.13(a) is the average BLER among the same 10 users under the EESM algorithm. Clearly, BLER under EESM exceeds (violates) the 10% target and increases with λ.

In Fig. 5.13(b), we plot the average sum of throughput of the 10 eMBB users under DELUXE over $5,000$ TTIs. Each point in the figure is the sum of each of the 10 eMBB user's



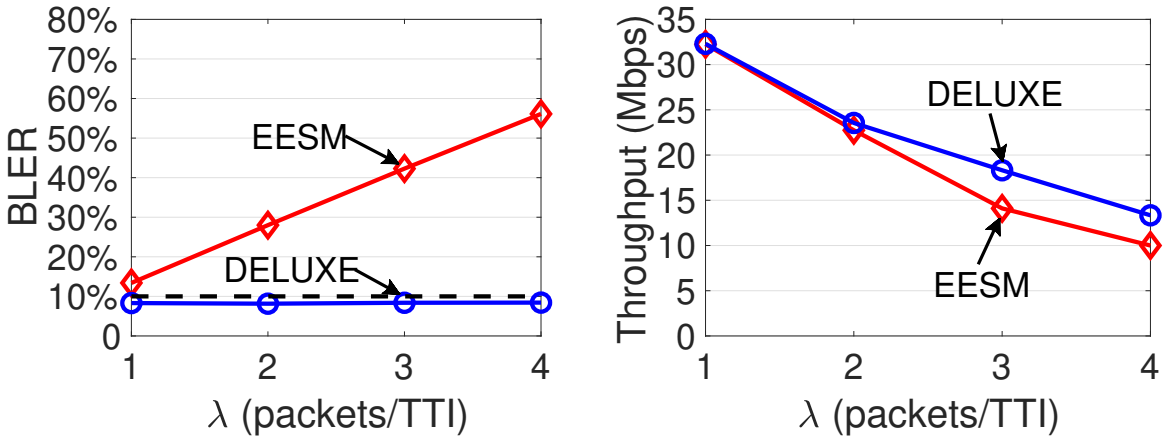
(a) BLER vs. λ (b) Sum throughput vs. λ

Figure 5.13: BLER and sum throughput performance under varying λ under DELUXE and EESM. Multi-path channel model is CDL-C.

average throughput over the $5,000$ TTIs. We can see the sum of eMBB throughput under DELUXE decreases as λ increases. This is because the impact of preemptive puncturing on eMBB is more profound as λ increases. Also shown in Fig. 5.13(b) is the sum eMBB throughput under EESM. We see that DELUXE's sum eMBB throughput is comparable to EESM, which is consistent to what we have seen in Fig. 5.11(a).

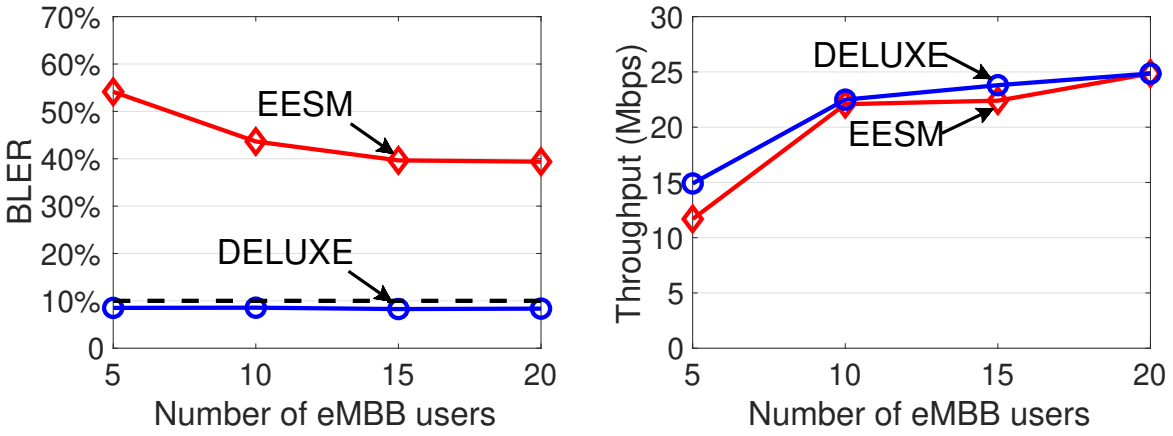
Varying Channel Models We now change the multi-path channel model in the last experiment from CDL-C to TDL-C repeat the same process (i.e., examining DELUXE's performance for varying λ). In Fig. 5.14(a), we find that average BLER under DELUXE still stays below the $\gamma = 10\%$ target, despite a change of channel model. Also shown in Fig. 5.14(a) is the average BLERs under EESM. Comparing to that in Fig. 5.13(a) for EESM, the average BLER performs better under TDL-C model. However, it still exceeds (violates) the 10% target threshold significantly when $\lambda \geq 2$.

In Fig. 5.14(b), we plot the sum of average throughput of the 10 eMBB users under DELUXE. The observations are similar to that for Fig. 5.13(b) and we omit to repeat the discussion.



(a) BLER vs. λ (b) Sum throughput vs. λ

Figure 5.14: BLER and sum throughput performance under varying λ under DELUXE and EESM. Multi-path channel model is TDL-C.



(a) BLER vs. number of eMBB users (b) Sum throughput vs. number of eMBB users

Figure 5.15: BLER and sum throughput performance under varying number of eMBB users under DELUXE and EESM.

Varying Number of Users We vary the number of eMBB users $|\mathcal{X}|$ in the cell and evaluate the performance of DELUXE. We assume CDL-C mult-path channel model and $\lambda = 2$. In Fig. 5.15(a), we plot the average BLER of DELUXE over $5,000$ TTIs under varying number of eMBB users. We see that DELUXE successfully stays below the 10% BLER target while the average BLER of EESM exceeds (violates) the 10% threshold substantially. In Fig. 5.15(b), we show the sum of average throughput over the $5,000$ TTIs achieved by DELUXE. Again, DELUXE is able to achieve comparable throughput to that by the EESM scheme under different number of users. Note that the throughput of DELUXE increases with the number of users. This is because as the number of user increases, the average number of RBs allocated to each user decreases. Since each user can only choose one MCS for all its allocated RBs, the fewer number of RBs allocated to him, the few number of joint channel conditions needs to be considered. Therefore, with fewer number of RBs per users, it is more likely to choose a higher MCS level for a user (under a random RB allocation).

Execution Time of DELUXE We now investigate the timing performance of DELUXE for MCS selection. Note that the execution time of DELUXE is independent of all other link and PHY layer components (i.e., RB allocation, modulation/demodulation, coding/decoding, among others). So instead of running the full link-level NR simulator (for all link and PHY layer components) as in the previous experiments (which is very time consuming), it is sufficient to run only the DELUXE component.

Recall in Section 5.8, the execution time of DELUXE only depends on the size of input data L and the number of eMBB transmissions $|\mathcal{E}(t)|$ in a TTI t. Since $L = 100$ is fixed, we vary the number of eMBB transmissions per TTI and study the timing performance of DELUXE. For a given number of eMBB transmissions per TTI, we randomly generate the input data $\tilde{\boldsymbol{S}}(t)$ to feed DELUXE (see Fig. 5.9).

Table 5.4 shows the execution time of DELUXE under different numbers of eMBB trans-

Table 5.4: Execution time of DELUXE. Results are averaged over 10^5 TTIs.

# of transmissions	10	20	30	40	50
Time (μs)	90.8	91.2	103.5	104.2	107.5

missions per TTI (10 to 50). In all cases, DELUXE's execution time is under 125 μs and meets the real-time requirement in 5G NR (under Numerology 3).

5.10 Chapter Summary

In this chapter, we investigated dynamic multiplexing of eMBB and URLLC through preemptive URLLC puncturing. We studied the problem of maximizing the throughput of eMBB (through MCS selection) while ensuring link reliability meets target requirement. We proposed DELUXE – a DL-based solution to the eMBB MCS selection under URLLC puncturing. The design of DELUXE includes: (i) a novel mapping to translate high dimensional information (channel condition and URLLC puncturing) on an eMBB code block into a low-dimensional representation with minimal information loss, (ii) a learning method to learn and predict the BLERs of an eMBB transmission under each MCS level based on the decoding results of eMBB training data, (iii) a fast calibration mechanism to offset potential prediction error based on recent decoding results, and (iv) a hybrid CPU and GPU-based implementation of DELUXE that can meet the real-time requirement in 5G NR. We implemented DELUXE on a 5G NR link-level simulator. Through extensive experimental results, we found that DELUXE can successfully maintain the desired link reliability for eMBB services while EESM fails to do so. Further, achieved throughput by DELUXE (via MCS selection) is comparable to EESM, which is considered the most spectral efficient LA approach for 4G LTE and 5G NR.

Chapter 6

Summary and Future Work

6.1 Summary

This dissertation studied real-time solutions to complex resource optimization problems in wireless networks. Such real-time requirements may originate from the short coherence time of wireless channels, the small time resolution in a radio frame structure, and low latency constraints from delay-sensitive applications. The objective of this dissertation is three-fold: i) We focus on resource optimization problems that arise from industry, ii) We pursue solutions that can achieve optimal or near-optimal objectives, and iii) Our solutions must be able to meet real-time requirements in the system.

This dissertation focuses on two types of resource optimization problems: i) problems that can be modeled with mathematical programs, and ii) problems that cannot be modeled with mathematical programs. For the first type of problems, we propose a novel approach that consists of: i) problem decomposition, ii) search intensification, and iii) GPU-based parallel processing. For the second type of problems, we propose to employ model-free DL or DRL techniques, along with judicious consideration of timing requirement throughout the design. Among the four main chapters in this dissertation, chapters 2 and 3 are devoted to the first type of problems, while chapters 4 and 5 address the second type of problems. A brief summary of these four chapters is given below:

- In Chapter 2, we investigated resource optimization for LTE's coexistence with Wi-Fi in unlicensed spectrum. Our optimization problem involves the division of transmission time for LTE and Wi-Fi on each channel and the allocation of resources during LTE's "on" periods. The objective is to minimize the adverse impact of LTE on Wi-Fi across all channels. The real-time requirement for finding an optimal solution is on ∼1 ms time scale. We developed a real-time algorithm to solve this problem, which we called CURT. CURT decomposes the original problem into a large number of small sub-problems and employs massive GPU cores to solve the sub-problems in parallel. By implementing CURT on NVIDIA GPU with CUDA, we showed that CURT can consistently find near-optimal solutions and meet the ∼1 ms real-time requirement.

- Chapter 3 studied the real-time PF-based resource optimization for 5G NR. The real-time requirement for determining an optimal PF scheduling solution is 125 μs (under 5G numerology 3), which is about ten times shorter than that in 4G LTE. To address this challenge, we proposed GPF+, which first breaks the PF scheduling problem into a large number of small sub-problems, and then selects a subset of sub-problems from the most promising search sub-space. In particular, we employed a cross-entropy based search intensification to identify the promising search sub-space. We implemented GPF+ using an NVIDIA Tesla v100 GPU. Experimental results demonstrated that GPF+ can find near-optimal scheduling solution within the ∼100 μs level timing constraint.

- Chapter 4 investigated the dynamic multiplexing of eMBB and URLLC services on the same channel. We studied the optimal placement of URLLC preemptive puncturing across eMBB resource allocations with the objective of minimizing the adverse impact on eMBB performance. The real-time requirement for determining the optimal URLLC puncturing solution is 1 ms (under 5G numerology 0). A major challenge in developing

a solution to this problem is that the objective function of the problem cannot be mathematically modeled in closed-form. To address this challenge, we proposed a model-free deep reinforcement learning based solution design, which we call DEMUX. DEMUX employs deep function approximators for learning an optimal algorithm for determining URLLC puncturing solutions. By implementing DEMUX in a link-level NR simulator, we showed that DEMUX significantly outperforms heuristics in the literature in terms of eMBB performance and can meet the 1 ms real-time requirement.

- In Chapter 5, we investigated an eMBB MCS selection problem under URLLC preemptive puncturing. The objective is to select an optimal MCS for each eMBB transmission so that it can meet a given BLER target against the interference from URLLC traffic. The real-time requirement for determining the optimal MCSs is 125 μs (under 5G numerology 3). Due to the lack of a closed-form mathematical model, this problem cannot be addressed by traditional optimization techniques. Our contribution in this chapter is the design of a DL based MCS selection method for eMBB services, which we call "DELUXE". The key idea behind DELUXE is to exploit DL to learn a neural network that can predict the BLERs of an eMBB transmission under each MCS level. Then an optimal MCS can be selected based on the predicted BLERs. DELUXE also includes a novel mapping to transform the original high-dimensional information into a low-dimensional representation, and a fast calibration mechanism to mitigate the BLER prediction errors. Link-level experimental results showed that DELUXE can successfully maintain the eMBB link reliability and spectral efficiency while meeting the $\sim$100 μs real-time requirement.

6.2 Future Work

There is a number of research opportunities that I have identified during my investigation. These opportunities are list below and deserve further research.

- **Search Intensification based on Machine Learning** As discussed in Chapter 3, an important step in our proposed solution is to identify a promising search sub-space within the original problem space through intensification. A limitation of the proposed cross-entropy based intensification in Chapter 3 is that we must develop a closed-form mathematical model for the resource optimization problem. This intensification approach cannot be applied to problems that do not have a closed-form mathematical program. To address this limitation, one may leverage model-free machine learning techniques. For example, we could employ deep neural networks to learn the unknown objective function of an optimization problem that has no closed-form model, and use it as a component of the intensification algorithm. The learned neural network can be used to identify the problem sub-space that is promising to offer optimal or near-optimal solutions. A major challenge in this approach is that the neural network based intensification must meet the real-time requirements in wireless networks.

- **Branch-and-Bound based Solution Design with GPU** Our GPU-based solutions in chapters 2 and 3 are heuristic algorithms that cannot provide a provable guarantee on the achievable optimality. To pursue real-time solutions with provable optimality gaps, one may combine GPU techniques with Branch-and-Bound (BB) algorithm, which is the most-common method for finding optimal or near-optimal solutions to discrete optimization problems. The basic idea of BB is to recursively divide the search space into smaller sub-spaces and eliminate sub-spaces that do not contain an optimal solution based on lower/upper bounds. A major obstacle in applying BB for

solving optimization problems in wireless networks is that its computation time is usually orders of magnitude longer than real-time requirements. A research topic is to investigate how to combine GPU parallel processing with BB methods to provide provably near-optimal solutions in real-time. For example, the bounding techniques in BB methods can be used for identifying promising problem sub-space with a provable guarantee on optimality gap. Then near-optimal solutions can be found in real-time via sampling similar to our design in Chapter 3. On the other hand, the branching steps in BB methods can be re-designed to better fit the GPU architecture. One possible approach is to include random sampling in the selection of sub-problems in each branching step, which can help reduce computational time significantly.

- **Hybrid CPU/GPU Parallel Algorithm Design** The essence of our solutions proposed in chapters 2 and 3 is problem decomposition and large-scale parallel processing using GPU cores. Our approach does not make sufficient use of CPU multi-processors. Modern CPU chips usually consist of multi-processors that can execute concurrent computation tasks. The CPU processors are optimized for sequential processing and can achieve a much lower latency compared to GPU processors. By fully exploiting computing powers across both CPU and GPU, it may be possible to enhance the performance of our real-time solution design. For example, we could decompose an optimization problem into sub-problems of different sizes. Then the smaller and simpler sub-problems can be sent to GPU for processing, while the larger and more complex ones are solved by CPU cores. This can achieve a higher level of concurrency and further reduce the processing latency. The challenge here is that the number of CPU cores are much fewer than that of GPU cores. So a research topic is to study how to decompose a problem into different sizes and a number that is suitable for a hybrid CPU/GPU computation.

Bibliography

[1] T. S. Rappaport, *Wireless Communications: Princlples and Practice.* Upper Saddle River, NJ, USA: Prentice-Hall, 1996.

[2] 3GPP TS 38.211 version 15.5.0, "NR; Physical channels and modulation," available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3213

[3] Qualcomm Research, "LTE in Unlicensed Spectrum: Harmonious Coexistence with Wi-Fi." Available: https://www.qualcomm.com/documents/lte-unlicensed-coexistence-whitepaper

[4] LTE-U Forum, "LTE-U CSAT Procedure TS Version 1.0." Available: http://www.lteuforum.org/uploads/3/5/6/8/3568127/lte-u_forum_lte-u_sdl_csat_procedure_ts_v1.0.pdf

[5] 3GPP TR 36.889 version 13.0.0, "Study on Licensed-Assisted Access to Unlicensed Spectrum." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2579

[6] 3GPP TS 36.211 version 15.2.0, "Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2425

[7] 3GPP TS 36.213 version 15.2.0, "Evolved Universal Terrestrial Radio Access (E-UTRA);

Physical Layer Procedures." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2427

[8] V. V. Vazirani, *Approximation Algorithms.* New York, NY, USA: Springer, 2003.

[9] Z. Guan and T. Melodia, "CU-LTE: Spectrally-Efficient and Fair Coexistence Between LTE and Wi-Fi in Unlicensed Bands," in *Proc. IEEE INFOCOM*, pp. 1-9, San Francisco, CA, USA, Apr. 2016.

[10] Y. Huang, Y. Chen, Y.T. Hou, W. Lou, and J.H. Reed, "Recent Advances of LTE/Wi-Fi Coexistence in Unlicensed Spectrum," *IEEE Network*, vol. 32, no. 2, pp. 107-113, Mar.-Apr. 2018.

[11] C. Cano, D.J. Leith, A. Garcia-Saavedra, and P. Serrano, "Fair Coexistence of Scheduled and Random Access Wireless Networks: Unlicensed LTE/WiFi," *IEEE/ACM Trans. Networking,* vol. 25, no. 6, pp. 3267-3281, Dec. 2017.

[12] A. Abdelfattah and N. Malouch, "Modeling and Performance Analysis of Wi-Fi Networks Coexisting with LTE-U," in *Proc. IEEE INFOCOM*, Atlanta, GA, USA, May. 2017.

[13] A. M. Voicu, L. Simic, and M. Petrova, "Inter-Technology Coexistence in a Spectrum Commons: A Case Study of Wi-Fi and LTE in the 5-GHz Unlicensed Band," *IEEE J. Sel. Areas Commun.,* vol. 34, no. 11, pp. 3062-3077, Nov. 2016.

[14] Q. Chen, G. Yu, R. Yin, A. Maaref, G. Y. Li, and A. Huang, "Energy Efficiency Optimization in Licensed-Assisted Access," *IEEE J. Sel. Areas Commun.,* vol. 34, no. 4, pp. 723-734, Apr. 2016.

[15] Q. Chen, G. Yu, and Z. Ding, "Optimizing Unlicensed Spectrum Sharing for LTE-U and

WiFi Network Coexistence," *IEEE J. Sel. Areas Commun.,* vol. 34, no. 10, pp. 2562-2574, Oct. 2016.

[16] H. Ko, J. Lee, and S. Pack, "Joint Optimization of Channel Selection and Frame Scheduling for Coexistence of LTE and WLAN," *IEEE Trans. Veh. Technol.,* vol. 67, no. 7, pp. 6481-6491, Jul. 2018.

[17] "T-Mobile Completes Nation's First Live Commercial Network Test of License Assisted Access (LAA)." Available: https://newsroom.t-mobile.com/news-and-blogs/lte-u.htm

[18] 3GPP TS 36.300 version 15.2.0, "Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (EUTRAN); Overall Description; Stage 2." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2430

[19] 3GPP TR 36.808 version 10.1.0, "Evolved Universal Terrestrial Radio Access (E-UTRA); Carrier Aggregation; Base Station (BS) Radio Transmission and Reception." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2487

[20] G. Bianchi and I. Tinnirello, "Kalman Filter Estimation of the Number of Competing Terminals in an IEEE 802.11 Network," in *Proc. IEEE INFOCOM,* pp. 844-852, San Francisco, CA, USA, Mar. 2003.

[21] A.L. Toledo, T. Vercauteren, and X. Wang, "Adaptive Optimization of IEEE 802.11 DCF Based on Bayesian Estimation of the Number of Competing Terminals," *IEEE Trans. Mobile Comput.,* vol. 5, no. 9, pp. 1283-1296, Sep. 2006.

[22] F. Capozzi, G. Piro, L.A. Grieco, G. Bogga, and P. Camarda, "Downlink Packet

Scheduling in LTE Cellular Networks: Key Design Issues and a Survey," *IEEE Commun. Surveys & Tutorials,* vol. 15, no. 2, pp. 678-700, 2013.

[23] G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization.* New York, NY, USA: John Wiley & Sons, 1999.

[24] Y.T. Hou, Y. Shi, and H.D. Sherali, *Applied Optimization Methods for Wireless Networks,* Chapter 5 and 6, Cambridge University Press, 2014.

[25] Nvidia, "CUDA Toolkit Documentation v9.2.148." Available: https://docs.nvidia.com/cuda/index.html

[26] Nvidia, "CUDA C Best Practices Guide." Available: https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html

[27] K. E. Baddour and N. C. Beaulieu, "Autoregressive Modeling for Fading Channel Simulation," *IEEE Trans. Wireless Commun.,* vol. 4, no. 4, pp. 1650-1662, Jul. 2005.

[28] Nvidia, "Data Sheet: Quadro P6000." Available: https://images.nvidia.com/content/pdf/quadro/data-sheets/192152-NV-DS-Quadro-P6000-US-12Sept-NV-FNL-WEB.pdf

[29] IBM ILOG CPLEX Optimizer, software available at http://www01.ibm.com/software/integration/optimization/cplex-optimizer

[30] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York, NY, USA: Freeman, 1990.

[31] Ericsson Technology Review, "5G new radio: Designing for the future." Available: https://www.ericsson.com/en/ericsson-technology-review/archive/2017/designing-for-the-future-the-5g-nr-physical-layer

[32] Qualcomm, "Making 5G NR a commercial reality," 2018, available: https://www.qualcomm.com/media/documents/files/the-3gpp-release-15-5g-nr-design.pdf

[33] Z. E. Ankarali, B. Peköz, and H. Arslan, "Flexible radio access beyond 5G: A future projection on waveform, numerology, and frame design principles," *IEEE Access,* vol. 5, pp. 18295-18309, May 2017.

[34] 3GPP TR 38.913 version 15.0.0, "Study on scenarios and requirements for next generation access technologies," available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2996

[35] 3GPP TR 38.804 version 14.0.0, "Study on New Radio access technology; Radio interface protocol aspects." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3070

[36] 3GPP TS 38.214 version 15.5.0, "NR; Physical layer procedures for data," available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3216

[37] 3GPP TS 38.101-1 version 15.0.0, "NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3201

[38] 3GPP TS 38.101-2 version 15.0.0, "NR; User Equipment (UE) radio transmission and reception; Part 2: Range 2 Standalone." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3284

[39] 3GPP TS 38.300 version 15.0.0, "NR; NR and NG-RAN overall description." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3191

[40] 3GPP TR 38.901 version 15.0.0, "Study on channel model for frequencies from 0.5 to 100 GHz." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3173

[41] 3GPP TS 36.101 version 15.1.0, "Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) radio transmission and reception." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2411

[42] 3GPP TR 22.891 version 14.2.0, "Feasibility study on new services and markets technology enablers; Stage 1." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2897

[43] T. S. Rappaport, *Wireless Communications: Principles and Practice.* Upper Saddle River, NJ, USA: Prentice-Hall, 1996.

[44] D. Schmalstieg and T. Höllerer, *Augmented Reality: Principles and Practice.* Reading, MA, USA: Addison-Wesley, 2016.

[45] G. Burdea, and P. Coiffet, *Virtual Reality Technology, 2nd ed.* New York, NY, USA: Wiley, 2003.

[46] L. Kong, M. K. Khan, F. Wu, G. Chen, and P. Zeng, "Millimeter-wave wireless communications for IoT-cloud supported autonomous vehicles: Overview, design, and challenges," *IEEE Commun. Mag.,* vol. 55, no. 1, pp. 62-68, Jan. 2017.

[47] S. Sesia, I. Toufik, and M. Baker, *LTE-The UMTS Long Term Evolution: From Theory to Practice.* New York, NY, USA: Wiley, 2009.

[48] O. Grondalen, A. Zanella, K. Mahmood, M. Carpin, J. Rasool, and O. Osterbo, "Scheduling policies in time and frequency domains for LTE downlink channel: a performance comparison," *IEEE Trans. Veh. Technol.,* vol. 66, no. 4, pp. 3345-3360, Apr. 2017.

[49] A. Stolyar, "On the asymptotic optimality of the gradient scheduling algorithm for multi-user throughput allocation," *Operations Research,* vol. 53, pp. 12-25, 2005.

[50] D. Tse, "Multiuser diversity in wireless networks: smart scheduling, dumb antennas and epidemic communication," in *IMA Workshop on Wireless Networks,* 2001. Available: https://web.stanford.edu/~dntse/papers/ima810.pdf

[51] R. Kwan, C. Leung, and J. Zhang, "Proportional fair multiuser scheduling in LTE," *IEEE Signal Process. Lett.,* vol. 16, pp. 461-464, June 2009.

[52] S. B. Lee, S. Choudhury, A. Khoshnevis, S. Xu, and S. Lu, "Downlink MIMO with frequency-domain packet scheduling for 3GPP LTE," in *Proc. IEEE INFOCOM,* pp. 1269-1277, Apr. 2009, Rio de Janeiro, Brazil.

[53] H. Zhang, N. Prasad, and S. Rangarajan, "MIMO downlink scheduling in LTE systems," in *Proc. IEEE INFOCOM,* pp. 2936-2940, Mar. 2012, Orlando, FL, USA.

[54] H. S. Liao, P. Y. Chen, and W. T. Chen, "An efficient downlink radio resource allocation with carrier aggregation in LTE-Advanced networks," *IEEE Trans. Mobile Computing,* vol. 13, no. 10, pp. 2229-2239, Oct. 2014.

[55] Y. Huang, S. Li, Y. T. Hou, and W. Lou, "GPF: A GPU-based design to achieve ~100 μs scheduling for 5G NR," in *Proc. ACM MobiCom,* pp. 207-222, Oct. 29-Nov. 2, 2018, New Delhi, India.

[56] C. Husmann, G. Georgis, K. Nikitopoulos, and K. Jamieson, "FlexCore: Massively parallel and flexible processing for large MIMO access points," in *Proc. of USENIX NSDI,* pp. 197–211, Mar. 2017, Boston, MA, USA.

[57] S. Roger, C. Ramiro, A. Gonzalez, V. Almenar, and A. M. Vidal, "Fully parallel GPU implementation of a fixed-complexity soft-output MIMO detector," *IEEE Trans. Veh. Technol.,* vol. 61, no. 8, pp. 3796-3800, Oct. 2012.

[58] Y. Chen, Y. Huang, C. Li, Y. T. Hou, and W. Lou, "Turbo-HB: A novel design and implementation to achieve ultra-fast hybrid beamforming" in *Proc. IEEE INFOCOM,* July 6–9, 2020, Toronto, Canada.

[59] K. Li, R. Sharan, Y. Chen, T. Goldstein, J. R. Cavallaro, and C. Studer, "Decentralized beamforming for massive MU-MIMO on a GPU cluster," in *Proc. IEEE GlobalSIP,* Dec. 2016, Washington, DC, USA.

[60] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: a GPU-accelerated software router," in *Proc. ACM SIGCOMM,* pp. 195-206, Aug. 2010, New Delhi, India.

[61] F. Fusco, M. Vlachos, X. Dimitropoulos, and L. Deri, "Indexing million of packets per second using GPUs," in *Proc. ACM SIGCOMM-IMC,* pp. 327-332, Oct. 2013, Barcelona, Spain.

[62] M. Varvello, R. Laufer, F. Zhang, and T. V. Lakshman, "Multilayer packet classification with graphics processing units," *IEEE Trans. Networking,* vol. 24, no. 5, pp. 2728-2741, Oct. 2016.

[63] Y. Zhao and F. Lau, "Implementation of decoders for LDPC block codes and LDPC convolutional codes based on GPUs," *IEEE Trans. Parallel Distrib. Syst.,* vol. 25, no. 3, pp. 663-672, Mar. 2014.

[64] A. Li, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Implementation of a fully-parallel turbo decoder on a general-purpose graphics processing unit," *IEEE Access,* vol. 4, pp. 5624-5639, Jun. 2016.

[65] Y. Huang, Y. Chen, Y. T. Hou, and W. Lou, "CURT: A real-time scheduling algorithm for coexistence of LTE and Wi-Fi in unlicensed spectrum," in *Proc. IEEE DySPAN,* pp. 1-9, Oct. 2018, Seoul, South Korea.

[66] C. Li, Y. Huang, Y. Chen, B. Jalaian, Y. T. Hou, and W. Lou, "Kronos: A 5G scheduler for AoI minimization under dynamic channel conditions," in *Proc. IEEE ICDCS,* pp. 1466-1475, Jul. 2019, Dallas, TX, USA.

[67] Qualcomm, "Making 5G NR a commercial reality," available: https://www.qualcomm.com/media/documents/files/whitepaper-making-5g-nr-a-reality.pdf

[68] 3GPP TS 36.213 version 15.0.0, "Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2427

[69] H. D. Sherali and W. P. Adams, *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems,* Chapter 8. Norwell, MA, USA: Kluwer Academic Publishers, 1999.

[70] F. Capozzi, G. Piro, L. Grieco, G. Boggia, and P. Camarda, "Downlink packet scheduling in LTE cellular networks: Key design issues and a survey," *IEEE Commun. Surveys Tutorials,* vol. 15, no. 2, pp. 678-700, 2013.

[71] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization.* New York, NY, USA: John Wiley & Sons, 1999.

[72] D. P. Bertsekas, *Dynamic Programming and Optimal Control.* Belmont, MA, USA: Athena Scientific, 1995.

[73] E. G. Talbi, *Metaheuristics: From Design to Implementation.* Hoboken, NJ, USA: Wiley, 2009.

[74] P. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Ann. Operations Res.,* vol. 134, no. 1, pp. 19-67, Feb 2005.

[75] R. Y. Rubinstein and D. P. Kroese, *The Cross-Entropy Method. A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning.* New York, NY, USA: Springer, 2004.

[76] Nvidia, "Optimizing parallel reduction in CUDA." Available: https://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf

[77] W. Yang, G. Durisi, and E. Riegler, "On the capacity of large-MIMO block-fading channels," *IEEE J. Sel. Areas Commun.,* vol. 31, no. 2, pp. 117-132, Feb. 2013.

[78] NVIDIA, "NVIDIA Tesla V100 GPU Architecture." Available: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[79] F. Zhang, J. Zhai, B. He, S. Zhang, and W. Chen, "Understanding co-running behaviors on integrated CPU/GPU architectures," *IEEE Trans. Parallel Distrib. Syst.,* vol. 28, no. 3, pp. 905-918, Mar. 2017.

[80] M. Daga, M. Nutter, and M. Meswani, "Efficient breadth-first search on a heterogeneous processor," in *Proc. IEEE Int. Conf. Big Data,* pp. 373-382, Oct. 2014, Washington DC, USA.

[81] ITU-R Report M.2410-0, "Minimum requirements related to technical performance for IMT-2020 radio interface(s)," available: https://www.itu.int/dms_pub/itu-r/opb/rep/R-REP-M.2410-2017-PDF-E.pdf

[82] 3GPP TR 38.824 version 16.0.0, "Study on physical layer enhancements for NR ultra-reliable and low latency case (URLLC)," available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3498

[83] Final Report of 3GPP TSG RAN WG1 Meeting #88 version 1.0.0, Feb. 13-17 2017, Athens, Greece, available: https://www.3gpp.org/ftp/TSG_RAN/WG1_RL1/TSGR1_88/Report/

[84] C. P. Li, J. Jiang, W. Chen, T. Ji, and J. Smee, "5G ultra-reliable and low-latency systems design," in *Proc. 2017 European Conference on Networks and Communications (EuCNC),* pp. 1-5, Jun. 2017, Oulu, Finland.

[85] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proc. Int. Conf. Learning Representations,* 2016.

[86] A. Anand, G. De Veciana, and S. Shakkottai, "Joint scheduling of URLLC and eMBB traffic in 5G wireless networks," in *Proc. IEEE INFOCOM,* pp. 1970-1978, Apr. 2018, Honolulu, HI, USA.

[87] Final Report of 3GPP TSG RAN WG1 Meeting #91 version 1.0.0, Nov. 27-Dec. 1 2017, Reno, NV, USA, available: https://www.3gpp.org/ftp/tsg_ran/WG1_RL1/TSGR1_91/Report/

[88] 3GPP TSG RAN WG1 Meeting #91, R1-1721452, Huawei, HiSilicon: Remaining aspects on pre-emption indication for DL multiplexing of URLLC and eMBB, Nov. 27-Dec. 1 2017, Reno, NV, USA.

[89] 3GPP TS 38.213 version 15.5.0, "NR; Physical layer procedures for control," available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3215

[90] P. Rybin, "On the error-correcting capabilities of low-complexity decoded irregular LDPC codes," in *Proc. IEEE Int. Symp. Inf. Theory,* pp. 3165-3169, Jun. 29- Jul. 4 2014, Honolulu, HI, USA.

[91] T. Richardson and S. Kudekar, "Design of low-density parity check codes for 5G new radio," *IEEE Commun. Mag.,* vol. 56, no. 3, pp. 28-34, Mar. 2018.

[92] A. K. Bairagi, M. S. Munir, M. Alsenwi, N. H. Tran, and C. S. Hong, "A matching based coexistence mechanism between eMBB and uRLLC in 5G wireless networks," in *Proc. ACM/SIGAPP Symp. Applied Computing 2019 (SAC '19),* pp. 2377-2384, Apr. 08-12, 2019, Limassol, Cyprus.

[93] J. Zhang, X. Xu, K. Zhang, B. Zhang, X. Tao, and P. Zhang, "Machine learning based flexible transmission time interval scheduling for eMBB and uRLLC coexistence scenario," *IEEE Access,* vol. 7, pp. 65811-65820, 2019.

[94] 3GPP TS 38.212 version 15.5.0, "NR; Multiplexing and channel coding," available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3214

[95] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning,* Cambridge, MA: MIT Press, 2016.

[96] NVIDIA, "cuDNN Developer Guide v7.6.3," 2019, available: https://docs.nvidia.com/deeplearning/sdk/cudnn-developer-guide/index.html

[97] S. Kullback, R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.,* vol. 22, no. 1, pp. 79-86, 1951.

[98] M. , J. Hu, S. Guo, and A. Y. Zomaya, "Distributed segment-based anomaly detection with Kullback-Leibler divergence in wireless sensor networks," *IEEE Trans. Inf. Forensics Security,* vol. 12, no. 1, pp. 101- 110, Jan. 2017.

[99] Y. Bu, S. Zou, Y. Liang, and V. V. Veeravalli, "Estimation of KL divergence: Optimal minimax rate," *IEEE Trans. Inf. Theory,* vol. 64, no. 4, pp. 2648-2674, Apr. 2018.

[100] O. Naparstek and K. Cohen, "Deep multi-user reinforcement learning for distributed dynamic spectrum access," *IEEE Trans. Wireless Commun.,* vol. 18, no. 1, pp. 310-323, Jan 2019.

[101] T. He, N. Zhao, and H. Yin, "Integrated networking, caching and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.,* vol. 67, no. 1, pp. 44-55, Jan. 2018.

[102] J. Zhu, Y. Song, D. Jiang, and H. Song, "A new deep-Q-learning-based transmission scheduling mechanism for the cognitive Internet of Things," *IEEE Internet Things J.,* vol. 5, no. 4, pp. 2375-2385, Aug. 2018.

[103] S. Ross, *A First Course in Probability,* 8th ed, Chapter 1, London, U.K.: Pearson, 2009.

[104] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. Int. Conf. Machine Learning,* 2014, pp. 387-395.

[105] MathWorks, "5G Toolbox Documentation," 2019, available: https://www.mathworks.com/help/5g/

[106] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467,* 2016.

[107] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learning Representations,* 2015.

[108] G. Ku and J. M. Walsh, "Resource allocation and link adaptation in LTE and LTE-advanced: A tutorial," *IEEE Commun. Surveys Tutorials,* vol. 17 , no. 3, pp. 1605-1633, Third Quarter 2015.

[109] J. Francis and N. Mehta, "EESM-Based Link Adaptation in Point-to-Point and Multi-Cell OFDM Systems: Modeling and Analysis," *IEEE Transactions on Wireless Communications,* vol. 13, no. 1, pp. 407-417, Jan. 2014.

[110] S. Simoens, S. Rouquette-Léveil, P. Sartori, Y. Blankenship, and B. Classon, "Error prediction for adaptive modulation and coding in multiple-antenna OFDM systems," *Signal Process.,* vol. 86, no. 8, pp. 1911-1919, Aug. 2006.

[111] T. L. Jensen, S. Kant, J. Wehinger, and B. H. Fleury, "Fast link adaptation for MIMO OFDM," *IEEE Trans. Veh. Technol.,* vol. 59, no. 8, pp. 3766-3778, Oct. 2010.

[112] R. C. Daniels, *Machine Learning for Link Adaptation in Wireless Networks,* Dissertation, UT Austin, Austin, TX, Dec. 2011. Available: http://hdl.handle.net/2152/ETD-UT-2011-12-4509

[113] Z. Dong, J. Shi, W. Wang, and X. Gao, "Machine learning based link adaptation

method for MIMO system," in *Proc. IEEE 29th Annu. Int. Symp. Pers. Indoor Mobile Radio Commun. (PIMRC),* pp. 1226-1231, Sep. 2018, Bologna, Italy.

[114] L. Zhang, J. Tan, Y.-C. Liang, G. Feng, and D. Niyato, "Deep reinforcement learning based modulation and coding scheme selection in cognitive heterogeneous networks," *IEEE Trans. Wireless Commun.,* vol. 18, no. 6, pp. 3281-3294, Jun. 2019.

[115] M. A. Albreem, M. Juntti, and S. Shahabuddin, "Massive MIMO detection techniques: A survey," *IEEE Commun. Surveys Tutorials,* vol. 21, no. 4, pp. 3109-3132, Fourth Quarter 2019.

[116] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," in *Workshop on Deep Learning, NIPS,* 2013.

[117] NVIDIA, "CUDA C++ programming guide v11.0.3." Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[118] Y. Huang, S. Li, C. Li, Y. T. Hou and W. Lou, "A deep-reinforcement-learning-based approach to dynamic eMBB/URLLC multiplexing in 5G NR," *IEEE Internet of Things Journal,* vol. 7, no. 7, pp. 6439-6456, Mar. 2020.

[119] K. I. Pedersen, G. Pocovi, and J. Steiner, "Preemptive scheduling of latency critical traffic and its impact on mobile broadband performance," in *Proc. IEEE 87th Veh. Technol. Conf.,* pp. 1–6, Jun. 2018, Porto, Portugal.

[120] T. Yu, X. Wang, and A. Shami, "UAV-enabled spatial data sampling in large-scale IoT systems using denoising autoencoder neural network," *IEEE Internet Things J.,* vol. 6, no. 2, pp. 1856-1865, Apr. 2019.

[121] H. He, C. Wen, S. Jin, and G. Y. Li, "Deep learning-based channel estimation for beamspace mmwave massive MIMO systems," *IEEE Wireless Commun. Lett.,* vol. 7, no. 5, pp. 852-855, Oct. 2018.

[122] M. Chen, W. Saad, C. Yin, and M. Debbah, "Echo state networks for proactive caching in cloud-based radio access networks with mobile users," *IEEE Trans. Wireless Commun.,* vol. 16, no. 6, pp. 3520-3535, Jun. 2017.

[123] J. Cui, Y. Liu, and A. Nallanathan, "Multi-agent reinforcement learningbased resource allocation for UAV networks," *IEEE Trans. Wireless Commun.,* vol. 19, no. 2, pp. 729-743, Feb. 2020.

[124] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Artificial neural networks-based machine learning for wireless networks: A tutorial," *IEEE Commun. Surveys Tutorials,* vol. 21, no. 4, pp. 3039-3071, Jul. 2019.

[125] R. Giuliano and F. Mazzenga, "Dimensioning of OFDM/OFDMA-based cellular networks using exponential effective SINR," *IEEE Trans. Veh. Technol.,* vol. 58, no. 8, pp. 4204-4213, Oct. 2009.

[126] S. Lagen, K. Wanuga, H. Elkotby, S. Goyal, N. Patriciello, and L. Giupponi, "New radio physical layer abstraction for system-level simulations of 5G networks," in *Proc. IEEE Int. Conf. Commun.,* Jun. 2020, pp. 1-7.

[127] T. M. Cover and J. A. Thomas, *Elements of Information Theory.* Hoboken, NJ, USA: Wiley, 2006.