Scheduling Batching Computing and Communication Tasks : Theoretical Foundation and Algorithm Design

Hehuan Shi

université
PARIS-SACLAY

Scheduling Batching Computing and Communication Tasks

Theoretical Foundation and Algorithm Design

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 580, sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat: Informatique
Unité de recherche: Université Paris-Saclay, CNRS, Laboratoire interdisciplinaire des sciences du numérique, 91405, Orsay, France
Référent: Faculté des sciences d'Orsay

Thèse présentée et soutenue à Paris-Saclay, le 29 Mars 2021, par

Hehuan SHI

Composition du jury:

Marco DI RENZO Professeur, Directeur de recherche, L2S CNRS, FRANCE	Président
Ken CHEN Professeur, Université Paris 13, FRANCE	Rapporteur & Examinateur
Francesco DE PELLEGRINI Professeur, Laboratoire Informatique d'Avignon, Université d'Avignon, FRANCE	Rapporteur & Examinateur
Tijani CHAHED Professeur, Telecom SudParis, FRANCE	Examinateur
Anastasios GIOVANIDIS Chargé de recherche, LIP6 CNRS, FRANCE	Examinateur
Lin CHEN Maître de Conférences (HDR), Université Paris-Saclay (LISN), FRANCE	Directeur de thèse

Thèse de doctorat

NNT: 2021UPASG025

Acknowledgements

First of all, I would like to extend my sincere gratitude to my supervisor, Prof. Lin Chen, who illuminates my way to science. Thank him for giving me the opportunity to study for my doctorate four years ago. Thank him for giving me the most patience. When I started my PhD, I was totally a beginner on doing research, on writing paper, especially on research field I have focused on. He has given me many valuable and pertinent suggestions, and spent loads of time and efforts on helping me revise my paper word by word. I have earned a lot from both on how to be a researcher and how to research. I am very proud to be one of the students of Prof. Lin Chen. For four years, I have believed that a good teacher influences his students for a lifetime. I feel extremely lucky and want to sincerely say thank you from the bottom of my heart.

My special thanks then go to my country which funds me to study for my doctorate in France.

I would also like to give my gratitude to all of my friends and colleagues both in China and France from whom I have benefited enormously. Particularly, I would like to thank Dr. Jihong Yu, Dr. Jia Liu, Dr. Shen Peng who helped me start living in France and offered me lots of suggestions on my research. I would like to thank Dr. Chuan Xu, Dr. Yanpei Liu, Dr. Yifei Zhu, Dr. Rongrong Zhang, Dr. Ruqi Huang, Dr. Yue Ma and Yangke Sun for sharing a lot of great moments in Paris. I would also like to thank Chuang Yu, Zejun Deng, Heng Li, Rupu Yang, Huan Xu and so on for sharing many wonderful basketball time with me and making my life rich and colorful. I would also like to thank Prof. Duyan Bi, Prof. Yuelei Xu, and Dr. Linyuan He who are my former supervisors or teachers and who have been very concerned about my study and life.

Last but not least, I would like to devote the most deep gratitude to each member of my family: my parents, my parents-in-law, my elder sister, and my wife for their unconditional love, support and encouragement. Especially, I want to express the most special gratitude to my wife Jing Wang. Thanks for being at my side for all these years.

Résumé

Dans cette thèse, nous formulons et analysons une classe de problèmes fondamentaux d'ordonnancement des tâches découlant d'une variété de systèmes informatiques et de communication émergents : les tâches sont divisées en groupes; celles d'un groupe peuvent être mises en lots et exécutées simultanément; l'objectif de l'ordonnanceur est de concevoir des algorithmes d'ordonnancement maximisant l'utilité globale du système. Sous le parapluie générique ci-dessus, nous étudions différentes classes de problèmes d'ordonnancement de tâches par lots, en établissant le cadre théorique correspondant, en concevant des algorithmes d'ordonnancement hors ligne et en ligne, et en illustrant leur application dans la planification des tâches de communication et d'informatique.

Nous commençons par le scénario de base de l'ordonnancement des tâches par lots. Il y a un ensemble de tâches à exécuter sur un certain nombre de machines. Certaines tâches peuvent être exécutées simultanément sur une seule machine, tandis que d'autres nécessitent l'utilisation exclusive d'une machine entière. Nous recherchons une politique d'ordonnancement optimale pour maximiser l'utilité globale du système. Nous développons un cadre algorithmique pour le problème d'ordonnancement ci-dessus dans la forme générique qui peut atteindre 1/2-optimality, surperformant le résultat le plus connu. La technicité de base de notre conception est un mécanisme de relaxation LP adapté et une approche d'arrondissement et de coloration qui transforme la solution de la relaxation LP en une politique d'ordonnancement optimale de 1/2. Nous démontrons ensuite l'application de notre cadre algorithmique pour résoudre le problème de diffusion proportionnelle généralisée en développant un algorithme d'approximation déterministe produisant une politique d'ordonnancement $l_{min}/(2(l_{min}+1))$-optimale, alors qu'il n'existe que des algorithmes randomisés dans la littérature.

Nous formulons et analysons ensuite un problème fondamental d'ordonnancement de transmission en liaison descendante dans les systèmes de communication sans fil, composé d'une station de base et d'un ensemble d'utilisateurs, chacun demandant un pa-

quet à servir dans une fenêtre de temps. Certains paquets sont demandés par plusieurs utilisateurs et peuvent être servis simultanément en raison de la nature de médium sans fil. Par rapport au modèle de base, il y a deux particularités. Premièrement, chaque demande peut être servie par un sous-ensemble de stratégies de transmission. Deuxièmement, les demandes doivent être servies de la manière FIFO. Nous recherchons un algorithme de planification de transmission en liaison descendante maximisant l'utilité globale du système. Nous établissons d'abord sa dureté en prouvant que (1) le problème hors ligne est NP-hard et (2) le problème en ligne est inapproximable dans sa forme générique. Compte tenu du résultat de dureté, nous développons ensuite des algorithmes d'approximation avec une garantie de performance mathématiquement prouvée en termes d'approximation et de rapports compétitifs pour les paramètres hors ligne et en ligne, respectivement.

La troisième contribution de cette thèse concerne l'ordonnancement des tâches par lots de ressources contiguës, qui est un problème bidimensionnel d'ordonnancement des tâches. Un ensemble de tâches doit être exécuté sur un pool de ressources continues, chacune nécessitant un certain temps et une ressource contiguë ; certaines tâches peuvent être exécutées simultanément en lot en partageant la ressource, tandis que d'autres nécessitent une utilisation exclusive de la ressource ; les tâches sont servies de la manière FIFO. Nous recherchons une allocation optimale des ressources et la politique d'ordonnancement connexe maximisant l'utilité globale du système. Ce problème se pose dans une variété de domaines de l'ingénierie, où les ressources de communication et de stockage sont des goulots d'étranglement potentiels et doivent donc être soigneusement programmés. Les deux problèmes précédents peuvent être considérés comme les cas dégénérés du problème d'ordonnancement des tâches de mise en lots des ressources contiguës. Deux exemples motivants sont le problème de liaison du spectre dans les systèmes d'accès au spectre et le problème d'allocation de stockage dynamique dans les systèmes informatiques. Nous fournissons une analyse algorithmique complète sur le problème en établissant sa dureté et en développant des algorithmes de programmation d'approximation pour les paramètres hors ligne et en ligne avec une garantie de performance éprouvée.

Abstract

In this thesis we formulate and analyze a class of fundamental task scheduling problems arising from a variety of emerging computing and communication systems: tasks are partitioned into groups; those in a group can be batched and executed simultaneously; the goal faced by the scheduler is to design scheduling algorithms maximizing the overall system utility. Under the above generic umbrella, we investigate different classes of batching task scheduling problems, establishing the corresponding theoretical framework, designing both offline and online scheduling algorithms, and illustrating their application in scheduling communication and computing tasks.

We start by the baseline scenario of batching task scheduling. There is a set of tasks to be executed on a number of machines. Some tasks can be executed simultaneously on a single machine, while others require exclusive use of an entire machine. We seek an optimal scheduling policy to maximize the overall system utility. We develop an algorithmic framework for the above scheduling problem in the generic form that can achieve 1/2-optimality, outperforming the best known result. The core technicality in our design is an adapted LP relaxation mechanism and a rounding and coloring approach that turns the solution of the LP relaxation to a 1/2-optimal feasible scheduling policy. We then demonstrate the application of our algorithmic framework to solve the generalized proportional broadcast problem by developing a deterministic approximation algorithm outputting an $l_{min}/(2(l_{min}+1))$-optimal scheduling policy, while there exist only randomized algorithms in the literature.

We then formulate and analyze a fundamental downlink transmission scheduling problem in wireless communication systems, composed of a base station and a set of users, each requesting a packet to be served within a time window. Some packets are requested by several users and can be served simultaneously due to the broadcast nature of the wireless medium. Compared to the baseline model, there are two particularities. First, each request can be served by a subset of transmission strategies. Second, requests need to be served in the FIFO manner. We seek a downlink transmission scheduling

algorithm maximizing the overall system utility. We develop an algorithmic framework of the formulated downlink data transmission scheduling problem in both offline and online settings. We first establish its hardness by proving that (1) the offline problem is NP-hard and (2) the online problem is inapproximable in its generic form. Given the hardness result, we then develop approximation algorithms with mathematically proven performance guarantee in terms of approximation and competitive ratios for the offline and online settings, respectively.

The third contribution of this thesis concerns the contiguous-resource batching task scheduling, which is a two-dimensional task scheduling problem. A set of tasks need to be executed on a pool of continuous resource, each requiring a certain amount of time and contiguous resource; some tasks can be executed simultaneously in batch by sharing the resource, while others requiring exclusive use of the resource; tasks are served in the FIFO manner. We seek an optimal resource allocation and the related scheduling policy maximizing the overall system utility. This problem arises in a variety of engineering fields, where communication and storage resources are potential bottlenecks and thus need to be carefully scheduled. The previous two problems can be regarded as the degenerated instances of the contiguous-resource batching task scheduling problem. Two motivating examples are the spectrum bonding problem in spectrum access systems and the dynamic storage allocation problem in computer systems. We deliver a comprehensive algorithmic analysis on the problem by establishing its hardness and developing approximation scheduling algorithms for both offline and online settings with proven performance guarantee.

Contents

1 Introduction **1**

1.1 Background and Motivation 1

1.2 Thesis Overview and Organization 2

2 Related Work **5**

2.1 Broadcast Scheduling 6

2.2 Task Scheduling 7

2.2.1 Task Scheduling without Batching 7

2.2.2 Task Scheduling with Batching 8

2.3 Downlink Transmission Scheduling 8

2.4 Bandwidth and Storage Allocation Problems 9

2.4.1 Bandwidth Allocation Problem 10

2.4.2 Storage Allocation Problem 11

2.5 Conclusion 11

3 Batching Task Scheduling: Baseline Algorithmic Framework **13**

3.1 Introduction 13

3.2 System Model and Problem Formulation 15

3.2.1 System Model 15

3.2.2 Problem Formulation 16

3.3 Our Algorithmic Framework 18

3.3.1 Task Graph Construction 18

3.3.2 From Task Scheduling to MWIS 20

3.3.3 LP Relaxation 20

3.3.4 Constructing a Feasible Scheduling Policy: Rounding and Coloring 22

3.3.5 Approximation Ratio Analysis 23

3.4 The Case of Unbounded Batching 24

3.4.1 Task Graph Construction 24
3.4.2 LP Relaxation 25
3.4.3 Approximation Algorithm 26
3.5 Applying Our Algorithmic Framework to Solve Generalized Proportional Broadcast Scheduling Problem 26
3.5.1 Problem Statement 27
3.5.2 Deterministic Approximation Scheduling Algorithm 29
3.5.3 Approximation Ratio Analysis 34
3.6 Numerical Analysis 35
3.6.1 Baseline Scenario of Batching Task Scheduling Problem 36
3.6.2 Proportional Broadcast Scheduling Problem 37
3.7 Conclusion 38
3.8 Appendix 39
3.8.1 Proof of Lemma 3.1 39
3.8.2 Proof of Lemma 3.2 40
3.8.3 Proof of Theorem 3.1 41
3.8.4 Proof of Lemma 3.3 42
3.8.5 Proof of Theorem 3.3 43
3.8.6 Proof of Lemma 3.4 44
3.8.7 Proof of Lemma 3.5 44
3.8.8 Proof of Theorem 3.4 45

4 Downlink Transmission Scheduling with Data Sharing 49
4.1 Introduction 50
4.2 System Model and Problem Formulation 51
4.2.1 System Model 51
4.2.2 Problem Formulation 53
4.3 The Offline Case 55
4.3.1 Problem Hardness 55
4.3.2 Request Graph 56
4.3.3 From Downlink Transmission Scheduling to Maximum Weighted Independent Set 60
4.3.4 LP Relaxation 61
4.3.5 Approximation Scheduling Algorithm Design 63
4.3.6 Performance Analysis 64

4.4 The Online Case . . . 65
4.4.1 Problem Inapproximability . . . 65
4.4.2 Online Scheduling Algorithm Design . . . 66
4.5 Numerical Analysis . . . 68
4.5.1 Scenario 1 . . . 69
4.5.2 Scenario 2 . . . 70
4.5.3 Scenario 3 . . . 71
4.6 Conclusion . . . 72
4.7 Appendix . . . 72
4.7.1 Integer Linear Problem Formulation of Offline Downlink Transmission Scheduling . . . 72
4.7.2 Proof of Theorem 4.1 . . . 73
4.7.3 Proof of Lemma 4.1 . . . 74
4.7.4 Proof of Lemma 4.2 . . . 76
4.7.5 Proof of Theorem 4.2 . . . 77
4.7.6 Proof of Theorem 4.4 . . . 78

5 Contiguous-resource Batching Task Scheduling 81
5.1 Introduction . . . 82
5.2 System Model . . . 83
5.3 The Offline Case . . . 87
5.3.1 Problem Formulation and Hardness . . . 87
5.3.2 Request Graph . . . 88
5.3.3 From Channel Bonding to Maximum Weighted Regular Independent Set . . . 92
5.3.4 LP Relaxation . . . 93
5.3.5 Approximation Scheduling Algorithm . . . 97
5.3.6 Performance Analysis . . . 100
5.4 The Online Case . . . 102
5.4.1 Inapproximability . . . 103
5.4.2 Online Scheduling Algorithm Design . . . 103
5.4.3 Performance Analysis . . . 106
5.5 Numerical Analysis . . . 107
5.5.1 Scenario 1 . . . 107
5.5.2 Scenario 2 . . . 108

5.5.3 Scenario 3 . . . 109
5.6 Conclusion and Perspective . . . 110
5.7 Appendix . . . 111
5.7.1 Integer Linear Problem Formulation of Offline Channel Bonding 111
5.7.2 Proof of Theorem 5.1 . . . 112
5.7.3 Proof of Lemma 5.1 . . . 112
5.7.4 Proof of Lemma 5.2 . . . 113
5.7.5 Proof of Lemma 5.3 . . . 115
5.7.6 Proof of Lemma 5.4 . . . 116
5.7.7 Proof of Lemma 5.5 . . . 118
5.7.8 Proof of Lemma 5.6 . . . 119
5.7.9 Proof of Theorem 5.2 . . . 119
5.7.10 Proof of Theorem 5.4 . . . 120

6 Conclusion and Prospective **123**
6.1 Thesis Summary . . . 123
6.2 Open Questions and Future Work . . . 124
6.2.1 Dependent batching task scheduling . . . 124
6.2.2 Flexible FIFO model . . . 125

Chapter 1

Introduction

This chapter provides a high-level description of the problems investigated in this thesis. It starts off by giving the background and motivation, then presenting the problems we address in this thesis, concluding with an overview of the organization of the thesis.

1.1 Background and Motivation

It is widely understood that communication and computing resources are potential bottlenecks in many emerging information systems. This is driven by the ever increasing number of communication and computing devices densely deployed today, which further underscores the necessity for efficient allocation and sharing of the common resources. Scheduling is the process of allocating a pool of resources among a set of tasks such that specific predefined objectives are achieved or optimized. Being an important branch in combinatorial optimization and theoretical computer science [25, 27, 29, 46], scheduling can be regarded as a decision-making process for optimizing one or more objectives.

In emerging computing and communication systems, the scheduling problems consider a set of tasks, each of which submits its request by requiring certain amount of specified resources for a specific time interval. The purpose of the problems is to optimally allocate limited resources to execute the tasks over time in order to achieve specific scheduling objectives, which can take many forms such as maximizing the system utility, minimizing makespan, minimizing the total delay cost, and so on [29]. More precisely, a schedule is a subset of the *Cartesian product* of three sets [46].

- A set of tasks (**What**) that should be executed.

- A set of time periods (**When**), intervals $[start, end]$ with $start \leq end$, where *start* and *end* are real-valued. Intuitively, one can associate with a task the time period during which it is executed.

- A set of resources that tasks occupy as they are executed. We denote this set as **Where**. It includes not only geographically localized resources, but also raw materials and other resources.

A schedule is a subset of **What**×**When**×**Where**. Intuitively, it specifies the resources that each task needs and when it needs them such that specific predefined objectives are achieved or optimized.

Unfortunately, most of scheduling problems are proven to be strongly NP-hard. Thus, unless P=NP, there are no efficient algorithms to find optimal scheduling policy for such problems, where an efficient algorithm is one that runs in time bounded by a polynomial in its input size [47]. A common approach for solving this problem is to relax the requirement of polynomial-time solvability. We try to find a scheduling policy that closely approximates the optimal scheduling policy in terms of system utility, but the goal is to relax the requirement as little as we possibly can.

In this thesis, we formulate and analyze a class of fundamental scheduling problems arising from a variety of emerging computing and communication systems, where tasks to be scheduled are partitioned into groups, and tasks in the same group can be batched and executed simultaneously. The goal is to design scheduling algorithms that maximize the system utility. Each of the problems, which is proven to be NP-hard, has its own particularities and calls for specific analysis that cannot draw upon existing results. For the considered problems, we consider *approximation algorithms* with proved performance guarantee.

1.2 Thesis Overview and Organization

In this section, we provide a high-level overview of our thesis. We first review the related literature in Chapter 2, and then describe the contributions of the thesis, which are presented sequentially in Chapters 3-5. Specifically, we start by investigating the baseline scenario of batching task scheduling problem in Section 3, and then we formulate and analyze the problems of downlink transmission scheduling with data sharing and contiguous-resource batching task scheduling, which are significant varieties

and extensions of the baseline scenario of batching task scheduling problem, in Sections 4 and 5, respectively. Figure 1.1 illustrates the structure of our thesis.

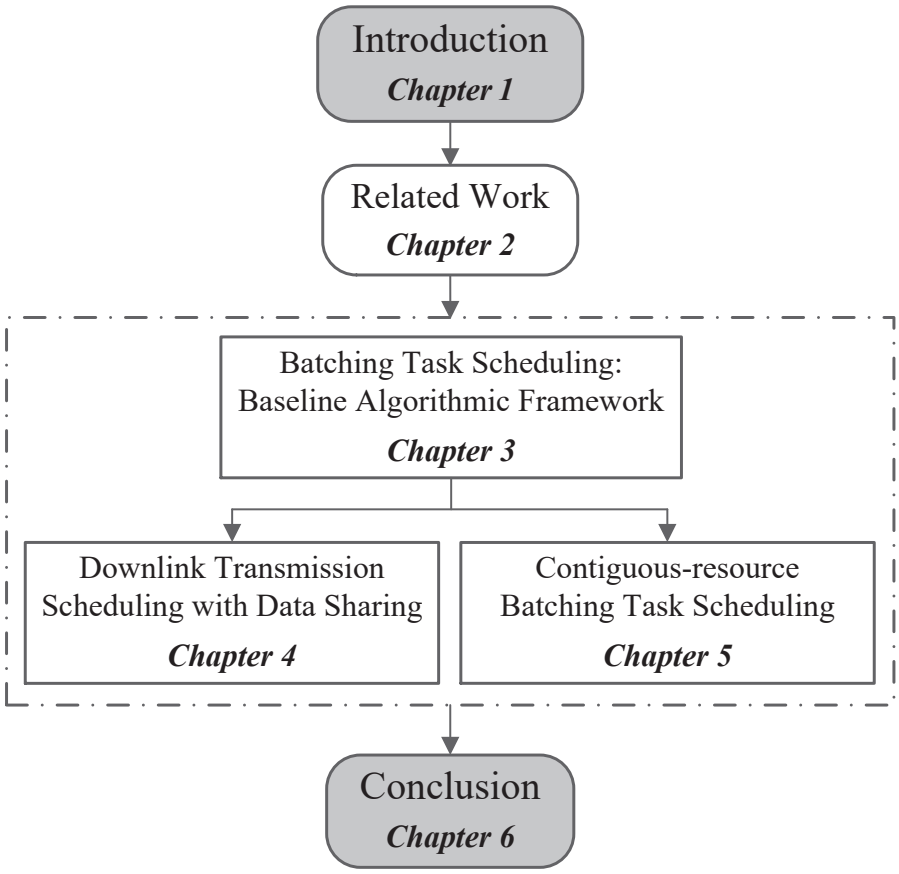


Figure 1.1: Thesis organization

Chapter 3: Batching Task Scheduling: Baseline Algorithmic Framework. This chapter investigates the batching task scheduling problem in its baseline form and develops an algorithmic framework. There is a set of tasks to be executed on a number of machines. Some tasks can be executed simultaneously on a single machine, while others require exclusive use of an entire machine. We seek an optimal scheduling policy to maximize the overall system utility. We develop an algorithmic framework for the above scheduling problem in the generic form that can achieve 1/2-optimality, outperforming the best known result. The core technicality in our design is an adapted LP relaxation mechanism and a rounding and coloring approach that turns the solution of the LP relaxation to a 1/2-optimal feasible scheduling policy. We then demonstrate the application of our algorithmic framework to solve the generalized proportional broadcast problem by developing the first deterministic approximation algorithm outputting an $l_{min}/(2(l_{min}+1))$-optimal scheduling policy, while there exist only randomized algo-

rithms in the literature.

Chapter 4: Downlink transmission scheduling with data sharing. This chapter formulates and analyzes a fundamental downlink transmission scheduling problem in a wireless communication system, composed of a base station, a set of transmission strategies and a set of users, each requesting a packet to be served within a time window. Some packets are requested by several users and can be served simultaneously due to the broadcast nature of the wireless medium. Compared to the baseline scenario of batching task scheduling, there are two particularities. First, each request can be served by a subset of transmission strategies. Second, requests should be served in the FIFO model. We seek a downlink transmission scheduling algorithm maximizing the overall system utility. In this chapter, we develop an algorithmic framework of the formulated downlink data transmission scheduling problem in both offline and online settings. We first establish its hardness, and then develop approximation algorithms with mathematically proven performance guarantee in terms of approximation and competitive ratios for the offline and online settings, respectively.

Chapter 5: Contiguous-resource batching task scheduling. This chapter formulates and analyzes the contiguous-resource batching task scheduling problem: a set of tasks need to be executed on a pool of continuous resource, and each task requires a certain amount of time and contiguous resource; some tasks can be executed simultaneously in batch by sharing the resource, while others requiring exclusive use of the resource; tasks are served in the FIFO manner. We seek an optimal resource allocation scheduling policy maximizing the overall system utility. In this chapter, we investigate both offline and online scheduling settings. In both cases, we establish the problem hardness and develop approximation algorithms with proven performance guarantee.

Chapter 2

Related Work

This thesis investigates a class of fundamental batching scheduling problems. From the algorithmic perspective, the problems we address are mostly related to the broadcast scheduling problem. This chapter reviews the related literature by starting with the broadcast scheduling problem in Section 2.1. We then extend the broadcast scheduling problem to the task scheduling problem by giving the overview of the task scheduling problems with and without batching in Section 2.2. To better understand the research status of the downlink transmission scheduling with data sharing and contiguous-resource task scheduling problems we address in the thesis, we review the previous works on these two topics in Section 2.3 and Section 2.4, respectively. Section 2.5 concludes the chapter.

Before presenting the related work, we introduce the following definitions.

Definition 2.1 (Feasible Scheduling Policy)**.** *We call a scheduling policy feasible for a problem if the policy satisfies all constraints of the problem.*

Definition 2.2 (Offline Scheduling Problem)**.** *The offline scheduling problem seeks an optimal feasible scheduling policy for the offline scheduler with the full knowledge of request/task information.*

Definition 2.3 (Online Scheduling Problem)**.** *The online scheduling problem seeks an optimal feasible scheduling policy for the online scheduler with only the current scheduling backlog.*

In the analysis for the maximum problems, the standard definitions of approximation factor and competitive ratio, which are used to quantify the efficiency of any offline and online algorithm respectively, are provided as below.

Definition 2.4 (Approximation Factor). *An algorithm* Π *is said to have approximation factor* $\rho > 1$ *if, for any problem instance* σ*, the system utility generated by* Π*, denoted by* $V_\Pi(\sigma)$*, is within factor* ρ *of the maximum system utility* $OPT(\sigma)$*, i.e.,* $\rho \cdot V_\Pi(\sigma) \geq OPT(\sigma)$*. We say that* Π *is a* $1/\rho-$*approximation algorithm.*

Definition 2.5 (Competitive Ratio). *An online algorithm* Π *is said to have competitive ratio* $\rho > 1$ *if, for any problem instance* σ*, the system utility generated by* Π*, denoted by* $V_\Pi(\sigma)$*, satisfies* $\rho \cdot V_\Pi(\sigma) \geq OPT(\sigma)$*. We say that* Π *is a* $1/\rho-$*competitive algorithm.*

2.1 Broadcast Scheduling

In the broadcast scheduling problem, a server broadcasts unit-size packets to a set of users, each requesting one of the packets with a time window, during which the requested packet needs to be received. Multiple users requesting a same packet can be served simultaneously if the packet is broadcast in the overlapped interval of their time windows. The problem faced by the central scheduler is to design a scheduling policy maximizing the system utility.

The broadcast scheduling problem has been considered in several papers. Gandhi *et al.* [18] developed a randomized rounding approach for fractional vectors defined on the edge-sets of bipartite graphs and provided a 3/4-approximation for the problem where the profit of each task is a fixed value (i.e., the value does not change with the time), and provided a $(1 - 1/e)$-approximation for the case where each task has an arbitrary time-window (instead of an interval) in which it must be processed, via a natural LP relaxation. Im and Sviridenko [24] gave a randomized $(0.5 - 0.75/e)$-approximation algorithm that improved the previous 3/4-approximation. Bansal *et al.* [4] gave an algorithm with an approximation ratio of 6/5 without proof that was unfortunately wrong. Chekuri *et al.* [12] adapted the ideas of [18] to obtain a 3/4-approximation for profit maximization with unimodal profit functions and a $(1 - 1/e)$-approximation for the case where each task is associated with an arbitrary non-negative profit function. The broadcast scheduling problem can be regarded as a degenerated instance of each of our scheduling problems we address in this thesis.

2.2 Task Scheduling

2.2.1 Task Scheduling without Batching

Given a set of tasks, each of which is associated with a time window and, if admitted, needs to be executed within the window, the canonical task scheduling problem seeks a scheduling algorithm maximizing system utility, with the assumption that any pair of tasks cannot be executed simultaneously. This unbatching task scheduling problem has been considered in several related works [5, 7, 10, 42]. Spieksma [42] considered the interval scheduling problem on a single machine. In the problem, the possible instances of a task are given explicitly as a set of time intervals. The goal is to pick a set of maximum number of nonintersecting time intervals so that at most one interval from each set of task instances is picked. This problem can be viewed as the discrete version for a special case of our problem. Spieksma [42] proved that the problem, to which the MAX 3-SAT-3 problem can be cast, is MAX-SNP hard, and gave a 1/2-approximation algorithm, and showed that the integrality gap of a linear programming formulation for this problem approaches 1/2 as well. Chuzhoy *et al.* [14] provided a randomized $(1 - 1/e)$-approximation algorithm. Bar-Noy *et al.* [7] provided a deterministic approximation algorithm with approximation ratio 2 via LP that was implicit in the weighted case, and they also proved that the LP-based approximation algorithm achieved 1/3-approximation for contiguous input, where there are two instances of tasks so that one terminates within time slot t and the other starts within t. Berman and DasGupta [10] proposed combinatorial two-phase algorithm that achieves ratio 2. Bar-Noy *et al.* [5] gave a 1/2-approximation algorithm via *local ratio technique* for discrete input and a $1/(2 + \epsilon)$-approximation algorithm for continuous input in the weighted case with arbitrary profits. Consider that tasks can be executed in multiple machines. In the case of unrelated machines, where the profit and processing time of each task are not identical in different machines, Bar-Noy *et al.* [7] demonstrated a greedy 1/2-approximation algorithm for the unweighted, and gave a 1/3-approximation factor for discrete weighted input and 1/4 for continuous weighted input. Bar-Noy *et al.* [5] achieved 1/2-approximation for discrete input and $1/(2+\epsilon)$ for continuous input. Berman and DasGupta [10] also gave a 1/2-approximation algorithm. In the case of K identical machines, i.e., where the profit and processing time of each task are the same for all machines, Bar-Noy *et al.* [7] achieved $(1 - 1/(1 + 1/K)^K)$-approximation for discrete input and $(1 - 1/(1 + 1/(2K))^K)$-approximation for continuous weighted

input. A $(1-(K/(K+1))^K)$-approximation algorithm was also implicit in Berman and DasGupta [10].

2.2.2 Task Scheduling with Batching

The works presented in Section 2.2.1 study a special case of the scheduling problem, where any pair of tasks cannot be executed simultaneously. In the generic scheduling problem, tasks are organized in groups such that those in the same group can be executed simultaneously. We call the simultaneously executed tasks a batch. The goal is to find a scheduling with batching that maximizes the total weight of the scheduled tasks. The number of tasks that can be batched simultaneously can be either *bounded* or *unbounded*. Both cases of the scheduling problem with batching are strongly NP-hard. Bar-Noy *et al.* [6] gave a 1/4-approximation algorithm that was based on the local ratio technique for discrete time input instance and $1/(4+\epsilon)$-approximation for continuous time input instance in the bounded case. When the batch size is unbounded, the approximation factor for the algorithm provided by Bar-Noy *et al.* [6] was reduced to 2 and $(2+\epsilon)$ for discrete and continuous time inputs, respectively. Moreover, they have extended the algorithm to multiple machines and obtained the same approximation factor.

2.3 Downlink Transmission Scheduling

The downlink transmission scheduling problem has been extensively explored in the telecommunication literature. In a typical formulation in wireless communication systems, the scheduler schedules packet transmission based on their requests and quality of service (QoS) requirements. The objective of the scheduler is to maximize the system utility. Many of the downlink transmission scheduling algorithms can be regarded as "gradient-based" algorithms. The key idea is to select the transmission rate vector to maximize the projection onto the gradient of the total utility [1, 2, 22, 30, 43].

Delay is arguably the most common constraint in download scheduling. In this regard, Sandrasegaran *et al.* [38] proposed a delay-prioritized scheduling algorithm to support real-time traffic in the downlink 3GPP LTE system maximizing the system throughput. Neely [32] considered the problem of maximizing throughput with random packet arrivals and time-varying channel reliability, and designed a utility maximization algorithm that used explicit delay information from the head-of-line packet at each user.

More generically to address the general QoS requirements, Andrews *et al.* [3] proposed an efficient way to support QoS of multiple real-time users sharing a wireless channel, and developed a scheduling policy to maximize channel capacity. Ryu *et al.* [36] proposed an urgency and efficiency based wireless packet scheduling algorithm to maximize throughput satisfying the QoS requirement, where the algorithm used the time-utility function as a scheduling urgency factor and the relative status of the current channel to the average one as an efficiency indicator of radio resource usage. Song [41] investigated downlink data scheduling with QoS provisioning over multiple channels, which, from a network point of view, provided line flexibility and granularity for resource allocation, and proposed corresponding scheduling algorithms achieving the maximum aggregate network utility. Ramli *ea al.* [34] investigated the performance of packet scheduling algorithms developed for single carrier wireless systems from a real-time video streaming perspective, and provided an algorithm achieving a high system throughput and supporting a large number of users by considering user fairness.

In more sophisticated settings, Rubio *et al.* [35] formulated a general multi-objective optimization problem and presented an approach to solve the non-convex optimization problem in multiuser MIMO broadcast networks implementing simultaneous wireless information and power transfer. Eisen *et al.* [16] considered the design of optimal resource allocation policies in wireless communication systems, modeled as a functional optimization problem with stochastic constraints, to maximize system utility; deep neural networks was trained with a primal-dual method to learn the resource allocation policy and optimize the primal/dual variables.

2.4 Bandwidth and Storage Allocation Problems

Consider the following generic setting. There is a set of independent users accessing a common frequency band whose width is F. Each user i submits its bandwidth request in the form of a quadruple (a_i, d_i, f_i, l_i), where a_i is the arrival time of the request and d_i is the deadline before which the request needs to be served, f_i with $0 < f_i \leq F$ is the quantity of frequency band requested by i, and l_i is the number of slots the user requests to use the frequency band. If request i is served, a reward w_i is generated to the system.

The *Bandwidth Allocation Problem* (BAP) is to decide which requests to be served so as to maximize the total weight of total reward of served requests subject to the constraint that the total size of served requests at any time must not exceed capacity

F and the frequency band allocated any request does not need to be contiguous [13]. The *Storage Allocation Problem* (SAP) is a variant of BAP, in which there are three additional constraints: (1) the specific portion of the resource allocated to a request cannot change during the time interval of the request, and (2) the allocation of resource must be contiguous, and (3) the two spatial intervals allocated to any two accepted requests should be disjoint if the reservation periods of the two requests intersect in their interior [8]. In the above two problems, the frequency band allocated to any pair of requests should not overlap and $d_i - a_i + 1 = l_i$ for each request i.

2.4.1 Bandwidth Allocation Problem

The BAP has been extensively explored in the literature, however, without systematically taking into account spectrum reuse, a key difference compared to the problem we address in the Chapter 5. In different contexts, the BAP is termed differently, such as resource allocation [11], resource-constrained scheduling [44], call admission control [20], etc.. We refer the readers to these references for detailed description of the specific context and problem setting. Via LP rounding, Phillips *et al.* [33] obtained a $1/6-$approximation algorithm. By the local ratio technique, Bar-Noy *et al.* [5] provided a $1/4-$approximation algorithm. Using a different idea, Chen *et al.* [13] provided a $1/3-$approximation algorithm in the special case where the weight of each request i is $(d_i - a_i)f_i$. Bar-Yehuda *et al.* [8] presented a deterministic polynomial-time approximation algorithms, whose approximation factor was $(2 + 1/(e-1)) \approx 2.582$. Based on the LP relaxation, Calinescu *et al.* [11] showed that the BAP could be $(1/2 - \epsilon)-$approximated in polynomial time, which improved upon earlier approximation results. Darmann *et al.* [15] considered a special case, where the rewards for all requests are identical, i.e., the goal is to maximize the number of served requests. They gave a deterministic $(1/2 - \epsilon)-$approximation algorithm. Shachnai *et al.* [40] induced the flexible bandwidth allocation problem (FBAP), where each request consists of a minimal and a maximal resource requirement, for the duration of its execution, as well as a profit accrues per allocated unit of the resource. In FBAP, the goal is to assign the available resource to a subset of requests such that the total profit is maximized. They presented a $1/3-$approximation algorithm by adapting the local ratio technique for FBAP.

2.4.2 Storage Allocation Problem

A geometric expression of SAP is to interpret each request as an axis-aligned rectangle that has a fixed weight, a fixed size (height and length), and can be moved vertically but not horizontally. The goal is to pack a subset of non-overlapping rectangles in a rectangular frame of given size, so as to maximize the total weight of the chosen rectangles [13]. Phillips *et al.* developed a $1/35-$approximation algorithm for the SAP. Leonardi *et al.* [28] obtained a $1/12-$approximation algorithm. Bar-Nay *et al.* [5] gave an approximation algorithm that yielded an approximation factor of 7 based on the local ratio technique. Chen *et al.* [13] studied a special case where all resource requirements were multiple of $1/K$ for some integer $K \geq 1$. They provided a polynomial-time approximation algorithm for the special case with an approximation factor of $\frac{e}{e-1}$ by assuming that the maximum resource requirement of any request was $O(1/K)$. Bar-Yehuda *et al.* [9] gave a randomized $1/(2+\epsilon)-$approximation algorithm, along with a deterministic $(\frac{e-1}{2e-1}-\epsilon)-$approximation algorithm for any fixed $\epsilon > 0$. Mömke and Wiese [31] studied the generalized version of SAP, and presented a randomized LP-based approximation algorithm with expected performance ratio of $2+\epsilon$ for any $\epsilon > 0$. Shachnai *et al.* [40] introduced the flexible storage allocation problem (FSAP), which was a variation of FBAP, and presented a $1/(2+\epsilon)-$approximation algorithm.

2.5 Conclusion

In this chapter, firstly, we have defined the offline and online scheduling problems and provided the standard definitions of approximation factor and competitive ratio, which are used to quantify the efficiency of any offline and online respectively. Then, we have reviewed the previous works on the problems of broadcast scheduling, task scheduling, downlink transmission scheduling, and bandwidth and storage allocation. We have a more in-depth understanding of research status on the problems, based on which, the thesis investigates a class of batching task scheduling problems in its baseline form, and extends the baseline scenario of batching task scheduling problem to the problems of downlink transmission scheduling with sharing and contiguous-resource batching task scheduling. The thesis focuses on developing approximation algorithms with proved performance guarantee for the problems, which are proven to be NP-hard.

Chapter 3

Batching Task Scheduling: Baseline Algorithmic Framework

This chapter investigates the batching task scheduling problem in its baseline form, where there is a set of tasks to be executed on a number of machines, and some tasks can be executed simultaneously on a single machine, while others require exclusive use of an entire machine. The goal is to develop an optimal scheduling policy to maximize the overall system utility. We develop a baseline algorithmic framework for the above scheduling problem in the generic form that can achieve 1/2-optimality, outperforming the best known result [6]. The core technicality in our design is an adapted LP relaxation mechanism and a rounding and coloring approach that turns the solution of the LP relaxation to a 1/2-optimal feasible scheduling policy, while the best existing result is a 1/4-approximation algorithm. We then demonstrate the application of our algorithmic framework to solve the generalized proportional broadcast problem by developing a deterministic approximation algorithm outputting an $l_{min}/(2(l_{min}+1))$-optimal scheduling policy, while there exist only randomized algorithms in the literature.

3.1 Introduction

Consider the following canonical *broadcast scheduling* problem[1] [12, 18, 24]. A server broadcasts a set of unit-size packets to a set of users. Each user requests one of the packets with a time window, during which the requested packet needs to be received.

[1]Strictly speaking, the problem corresponds more to the multicast setting from a networking perspective. In this thesis, we stick to the term broadcast to be coherent to the related literature on this problem.

Multiple users can be served simultaneously if they request the same packet and the packet is broadcast in the overlapped interval of their time windows. The server obtains a unit reward for each served user. The problem faced by the server is to find an optimal scheduling algorithm maximizing her overall reward. This so-called *pull-based* broadcast scheduling problem has attracted significant research attention because of its neat formulation going far beyond the broadcast context and its profound algorithmic implication: it represents a class of scheduling problems concerning how to group requests over time so as to optimize or satisfy certain scheduling objective or constraint.

Another example fitting in the above formulation is the *lock scheduling* problem arising from concurrency control [45]. In modern software and computing systems, objects (e.g., data, memory) are usually shared and concurrently accessed by a large number of applications or transactions, termed as *tasks*. Ensuring consistency of the shared objects in this context is of fundamental importance. A widely used mechanism is to rely on a lock manager to efficiently schedule the object access to ensure both data correctness and system efficiency. More specifically, consider an object accessed by multiple tasks, a task can access the object only if it is granted a *lock*. There are two types of locks: shared and exclusive locks. Shared locks are granted to tasks that do not modify the object, e.g., reading a data item in a database, while exclusive locks are given to tasks modifying the object, e.g., writing or updating a data item. The object can be accessed simultaneously by multiple tasks with shared locks, but only one single task with exclusive object. Given a number of tasks, each requiring to access the object within a certain period of time, the problem of lock scheduling consists of granting locks to them to maximize system utility, e.g., maximizing the number of executed tasks.

Motivated by the above two examples, we formulate the following baseline scenario of batching task scheduling problem. There is a set of tasks to be executed on a number of machines. Some tasks can be executed simultaneously by a single machine, while others require exclusive use of an entire machine. We seek an optimum scheduling policy to maximize the overall system utility.

- From a theoretical point of view, the baseline scenario of batching task scheduling problem is a significant generalization of the broadcast and lock scheduling problems. By casting the tasks into the packet requests such that the tasks that can be executed simultaneously correspond to the requests concerning the same packet to be broadcast, our problem degenerates to the broadcast scheduling problem. By regarding the sharing of machines as shared locks, the task scheduling problem

readily degenerates to the lock scheduling problem.

- From a practical point of view, the baseline scenario of batching task scheduling problem we formulate arises in a variety of engineering fields where computing, communication, and storage resources are potential bottlenecks and thus need to be carefully scheduled.

In this chapter, we establish an algorithmic framework on the above task scheduling problem in its most generic form. Our algorithmic framework allows to develop scheduling policies with guaranteed performance bound in a variety of scheduling contexts. Methodologically, we start with the baseline scenario without any assumption on the system setting to build our algorithmic framework in its generic form. We then demonstrate how our algorithmic framework can be adapted to solve other batch scheduling problems by analyzing the *proportional broadcast scheduling* problem, where a portion of the utility is obtained by a task even if it is not executed in totality.

Our main results in this chapter can be summarized as below.

- For the baseline scenario of batching task scheduling, we develop a 1/2-approximation algorithm, while the best existing result is a 1/4-approximation algorithm.

- For the proportional broadcast scheduling problem, we develop the first deterministic scheduling algorithm with constant approximation factor, while in the literature, there exist only randomized algorithms providing average performance guarantees.

3.2 System Model and Problem Formulation

3.2.1 System Model

As stated in the Introduction, we put the canonical broadcast scheduling problem in a generic context by considering the following task scheduling problem. We have a system composed of a set $\mathcal{K}$ of K machines, indexed from 1 to K. A set $\mathcal{N}$ of tasks, indexed from 1 to $N \triangleq |\mathcal{N}|$, are submitted to the system. Each task i is submitted at time a_i, the *release time*, and should be finished by time d_i, the *deadline* or *due date*; $d_i - a_i + 1$ is called its *slackness*. Let $\mathcal{K}_i$ denote the set of machines, on each of which task i can be scheduled. For each machine $k \in \mathcal{K}_i$, we denote $l_{i,k}$ the execution time of task i on machine k. The tasks are divided into B non-overlapping groups, indexed

from 1 to B; the tasks in the same group can be executed simultaneously on a same machine; we use b_i ($1 \leq b_i \leq B$) to denote the group index of task i; let $\mathcal{N}_b \in \mathcal{N}$ denote the set of tasks belonging to group b. If a task i is successfully executed on machine k before its deadline, a reward $w_{i,k}$ is generated to the system. For the tasks that can be executed simultaneously on a same machine, we say that they form a *batch*. Let $\pi_k \leq N$ denote the maximum batch size that can be supported by each machine k, i.e., at most π_k tasks of a same group can be executed simultaneously on each machine k. The problem is called *bounded* if $\pi_k < N$ and *unbounded* otherwise. We seek an optimal scheduling policy to maximize the overall reward within a time horizon T. Table 3.1 lists the main notations in the chapter.

3.2.2 Problem Formulation

In practice, modern computing and communication systems operate on slotted time, where time is discretized into slots and scheduling is performed at the slot level. We thus concentrate ourselves on the discretized model, under which, by normalizing the slot duration to unit time, all the parameters and variables in the problem formulation are restricted to integers. In this case, our problem, denoted by **P1**, can be formulated as an integer linear programming (ILP) as below.

$$
\begin{aligned}
\textbf{P1:}\quad & \text{maximize} \sum_{i\in\mathcal{N}} \sum_{k\in\mathcal{K}_i} w_{i,k} \cdot x_{i,k} \\
& \text{subject to} \\
& t_{j,k} - t_{i,k} \geq l_{i,k}, \qquad \forall i,j \in \mathcal{N},\ b_i \neq b_j,\ x_{i,k} = x_{j,k} = 1,\ t_{i,k} < t_{j,k} \\
& \sum_{i\in\mathcal{N}, t_{i,k} \leq t \leq t_{i,k}+l_{i,k}-1} x_{i,k} \leq \pi_k, \quad \forall 0 \leq t \leq T,\ 1 \leq k \leq K \\
& \sum_{k\in\mathcal{K}_i} x_{i,k} \leq 1, \qquad \forall i \in \mathcal{N} \\
& a_i \leq t_{i,k} \leq d_i - l_{i,k} + 1, \qquad \forall i \in \mathcal{N},\ x_{i,k} = 1,\ k \in \mathcal{K}_i \\
& x_{i,k} \in \{0,1\}, \qquad \forall i \in \mathcal{N},\ k \in \mathcal{K}_i.
\end{aligned}
$$

where $x_{i,k}$ is the binary variable indicating whether task i is executed or not on machine k, and $t_{i,k}$ is the time to start executing task i on machine k if $x_{i,k} = 1$. The first constraint implies that if task i cannot be executed simultaneously with task j on a same machine, i.e., $b_i \neq b_j$, the allocated time to execute them cannot overlap; The

Table 3.1: Main notations

$\mathcal{N}$	task set
N	number of tasks, $N = \lvert\mathcal{N}\rvert$
T	time horizon in number of time slots
B	number of task groups
K	number of machines in the system
π_k	maximum batch size supported on machine k
π	$\max_{1\leq k\leq K} \pi_k$
$\mathcal{N}_b$	set of group b tasks
a_i	arriving time of task i
d_i	deadline of task i
b_i	group index of task i
$\mathcal{K}_i$	set of machines on which task i can be executed
$l_{i,k}$	number of slots task i needs to be executed on machine k
$w_{i,k}$	reward of task i if it it executed on machine k
$x_{i,k}$	binary variable indicating whether task i is executed on machine k
$t_{i,k}$	time to start executing task i on machine k
G	task graph $G = (\mathcal{V}, \mathcal{E})$
$\mathcal{V}_i$	set of vertices of task i
$\mathcal{V}_{i,k}$	set of vertices in $\mathcal{V}_i$ corresponding to machine k
$\mathcal{V}_{i,k}(t)$	set of vertices $v \in \mathcal{V}_{i,k}$ with $t_v \leq t \leq t_v + l_v - 1$
$\mathcal{V}_{i,k,r}$	set of vertices in $\mathcal{V}_i$ corresponding to sub-machine r on machine k
$\mathcal{V}_{i,k,r}(t)$	set of vertices $u \in \mathcal{V}_{i,k,r}$ with $t_u \leq t \leq t_u + l_u - 1$
t_v	start-time of the interval corresponding to vertex v
l_v	length of the interval corresponding to vertex v
w_v	weight of vertex v
$\delta_{i,k}$	$d_i - a_i - l_{i,k} + 2$
δ	$\sum_{i\in\mathcal{N}} \sum_{k\in\mathcal{K}_i} \delta_{i,k}$
y_v	binary variable indicating whether vertex v is selected
$q_{b,t,k}$	binary variable indicating whether a task of group b is executed at slot t on machine k
$q^*_{b,t,k}$	value of $q_{b,t,k}$ in the LP relaxation
y^*_v	value of y_v in the LP relaxation
$\hat{N}$	scaling factor. $\hat{N} = R\delta^{1+\epsilon}$ and $\delta^{1+\epsilon}$ for bounded and unbounded batching
$\mathcal{C}$	ordered set of $2\hat{N} - 1$ colors
$\mathcal{C}_{b,t,k}$	set of colors already used to color the vertices v with $t_v \leq t \leq t_v + l_v - 1$ corresponding to group b and machine k
$\hat{\mathcal{V}}$	set of vertices output by Algorithm 1

second constraint indicates that at most π_k tasks can be executed together at the same time on each machine k; The third constraint indicates that each task i is executed at most once; The fourth constraint is the time constraint to execute task i.

When $K = 1$ and $\pi_1 = N$, we can cast the Knapsack problem with integer weights to **P1**. It then follows from the NP-completeness of the Knapsack problem with integer weights [23] that **P1** is NP-complete.

3.3 Our Algorithmic Framework

Given the hardness of our problem, we naturally focus on developing approximate algorithm. At the high level, our idea is to construct a graph, termed as *task graph*, to capture the relationships between tasks and cast **P1** to the Maximum Weighted Independent Set (MWIS) problem [37][2] in the task graph. To solve the MWIS, we construct an LP relaxation and solve the relaxed linear programming (LP) problem. By exploiting the structural properties of the task graph, we develop a coloring algorithm to find an independent set (IS) of the task graph that can map to a feasible scheduling policy, where we then prove the policy to be 1/2-approximation of the optimal solution of **P1**. The *feasible scheduling policy* for the baseline scenario of batching task scheduling is defined as below, concisely termed as feasible policy.

Definition 3.1 (Feasible Scheduling Policy for the Baseline Scenario of Batching Task Scheduling)**.** *We call a scheduling policy feasible if*

- *each task is executed at most once;*
- *the time intervals for executing any pair of tasks from different groups do not overlap on a same machine;*
- *at most π_k tasks of a same group can be executed simultaneously on each machine k.*

3.3.1 Task Graph Construction

We divide each machine k into π_k *sub-machines*. At most one task is allowed to be executed on each sub-machine at the same time. We then construct an undirected

[2]An independent set of a graph is a set of vertices, no two of which are linked by an edge. The maximum weighted independent set is to find the independent set maximizing the sum of weights of the vertices in the independent set.

graph $G = (\mathcal{V}, \mathcal{E})$ capturing the relationships among tasks, termed as *task graph*, by defining its vertices and edges as below.

Vertices. Consider each task $i \in \mathcal{N}$. For each sub-machine r ($1 \leq r \leq \pi_k$) on each machine $k \in \mathcal{K}_i$, we create a vertex v for each time interval of length $l_{i,k}$ in the time interval $[a_i, d_i]$. We say that v covers i, and v corresponds to sub-machine r and machine k. If the scheduler decides to execute i on sub-machine r of machine k in the time interval corresponding to v, we say that i is *instantiated* by v. Let $\mathcal{V}_i$ denote the set of vertices of task i; let $\mathcal{V}_{i,k,r}$ denote the set of vertices in $\mathcal{V}_i$ corresponding to sub-machine r and machine k. For each vertex $v \in \mathcal{V}_{i,k,r}$, we define a weight w_v and assign w_v to $w_{i,k}$. Let t_v and l_v denote the left boundary (i.e., starting time) and the length of the time interval corresponding to vertex v.

Let $\pi \triangleq \max_{1 \leq k \leq K} \pi_k$. Denote $\delta_{i,k} \triangleq d_i - a_i - l_{i,k} + 2$ and $\delta \triangleq \sum_{i \in \mathcal{N}} \sum_{k \in \mathcal{K}_i} \delta_{i,k}$. For each task i, there are $\sum_{k \in \mathcal{K}_i} \pi_k \delta_{i,k}$ vertices in $\mathcal{V}_i$. We can thus upper-bound the number of vertices in G by $\pi\delta$.

Edges. The edges in G capture the relationship among tasks and the number of tasks in each batch. We distinguish two types of edges.

Intra-task edges. For each task i, we construct an edge between each pair of vertices in $\mathcal{V}_i$. The intra-task edges model the constraint that any task is executed at most once.

Inter-task edges. This type of edges are further classified into two sub-categories.

- **Inter-task edges characterizing conflicts among tasks.** For each machine k and each pair of tasks i and j that cannot be executed simultaneously, i.e., $b_i \neq b_j$, we construct an edge between any pair of vertices $v \in \mathcal{V}_i$ and $u \in \mathcal{V}_j$ if v and u correspond to machine k and if the time intervals corresponding to v and u overlap, indicating that instantiating task i by v and instantiating task j by u cannot both happen in any feasible scheduling policy.

- **Inter-task edges modeling bounded batching.** For each machine k, each sub-machine r and each pair of tasks i, j with $b_i = b_j$ and $k \in \mathcal{K}_i \cap \mathcal{K}_j$, we construct an edge between each pair of vertices $v \in \mathcal{V}_{i,k,r}$ and $u \in \mathcal{V}_{j,k,r}$ if the time intervals corresponding to v and u overlap, making it impossible to instantiate i by v and also j by u.

3.3.2 From Task Scheduling to MWIS

We cast **P1** to the MWIS problem [37] in G. An IS of a graph is a set of vertices, no two of which are linked by an edge. The MWIS problem is to find an IS maximizing the sum of weights of the vertices in the IS. By choosing an IS in G we mean to execute the tasks instantiated by the vertices in the IS. We first show that there exists a one-to-one mapping between an IS of G and a feasible scheduling policy. To make our analysis more stream-lined, we put all the proofs in Appendix 3.8 and give proof sketch in the main text.

Lemma 3.1. *Each feasible scheduling policy maps to an IS of G, and vice versa.*

Proof Sketch. The sufficiency proof consists of deriving a contradiction if there exists a feasible policy mapping to a subset of vertices in $\mathcal{V}$ containing two neighboring vertices. The necessity proof is based on the construction of edges in G, where each type of edges ensures a kind of constraints for the feasible scheduling policy. □

Lemma 3.1 immediately leads to the following corollary.

Corollary 3.1. ***P1*** *can be cast to the MWIS problem on G, whose ILP formulation is given below.*

$$\begin{aligned} &\max \sum_{v\in\mathcal{V}} w_v \cdot y_v \\ &s.t.\ \ y_u + y_v \leq 1, && \forall uv \in \mathcal{E} \\ &\quad\ y_v \in \{0,1\}, && \forall v \in \mathcal{V} \end{aligned}$$

where y_v is the binary variable indicating whether vertex v is selected in the IS.

3.3.3 LP Relaxation

Given the NP-hardness of the MWIS problem, we design an approximation algorithm by rounding the solution of LP relaxation of **P1**. To this end, we need to exploit the particular structure of our problem. This section is focused on the construction of the LP relaxation of **P1**.

We first replace the constraint $y_v \in \{0,1\}$ with $y_v \geq 0$. It is well-known that the LP relaxation of the MWIS problem suffers the so-called *half integer* effect due to the edge constraint [39]. To mitigate this effect, we use the following constraints to replace the constraint $y_u + y_v \leq 1$ such that (1) any feasible scheduling policy is still feasible

in the relaxed problem, (2) non-feasible scheduling policies are eliminated as many as possible to facilitate the rounding process and to ensure the quality of the rounded integer solution.

- First, in any feasible scheduling policy, each task is executed at most once, leading to the following constraints.

$$\sum_{v \in \mathcal{V}_i} y_v \leq 1, \quad i \in \mathcal{N}. \tag{3.2}$$

- Second, to model the resource constraint, we introduce a set of binary variable $q_{b,t,k}$, where $b \in [1, B], t \in [0, T]$ and $k \in [1, K]$, to indicate whether at least one task of group b is executed or not at slot t on machine k. In the LP relaxation we have $0 \leq q_{b,t,k} \leq 1$. Any pair of tasks from different groups cannot be executed simultaneously on each machine, leading to the following constraint.

$$\sum_{1 \leq b \leq B} q_{b,t,k} \leq 1, \quad \forall 0 \leq t \leq T,\ 1 \leq k \leq K. \tag{3.3}$$

- Third, consider each machine k and each of its sub-machines r. In any feasible scheduling policy, for each slot t and each group b, at most one task belonging to group b can be served on the sub-machine if $q_{b,t,k} = 1$. Therefore, the sum of y_v's of vertices $v \in \{\mathcal{V}_{i,k,r}(t)\}_{i \in \mathcal{N}_b}$, which are adjacent to each other, is upper-bounded by $q_{b,t,k}$, where $\mathcal{V}_{i,k,r}(t) \triangleq \{u : u \in \mathcal{V}_{i,k,r}, t_u \leq t \leq t_u + l_u + 1\}$. Mathematically we have the following constraint.

$$\sum_{i \in \mathcal{N}_b} \sum_{v \in \mathcal{V}_{i,k,r}(t)} y_v \leq q_{b,t,k}, \quad \forall 1 \leq k \leq K,\ 1 \leq r \leq \pi_k,\ 1 \leq b \leq B,\ 0 \leq t \leq T. \tag{3.4}$$

By combining the above analysis, we construct the following LP relaxation of the formulated MWIS problem, denoted by **P1'**.

$$\begin{aligned}
\mathbf{P1'}: \quad & \max \sum_{v \in \mathcal{V}} w_v \cdot x_v \\
& \text{s.t. } (3.2), (3.3), (3.4) \\
& \quad y_v \geq 0, && \forall v \in \mathcal{V} \\
& \quad q_{b,t,k} \geq 0, && \forall 1 \leq b \leq B,\ 0 \leq t \leq T,\ 1 \leq k \leq K.
\end{aligned}$$

There is no need to explicitly add the constraints $y_v \leq 1$ and $q_{b,t,k} \leq 1$ since they are implied by (3.2) and (3.3), respectively. It is easy to see that any feasible scheduling policy is a feasible solution of **P1**'. Hence, an optimal fractional solution of **P1**' is an upper bound of the utility of any optimal feasible scheduling policy.

3.3.4 Constructing a Feasible Scheduling Policy: Rounding and Coloring

We now present our approximation algorithm that rounds the solution of **P1'** and constructs a feasible scheduling policy. Our approach generalizes the rounding and coloring technique developed in [7] adapted to our context, addressing the following two constraints in our problem: (1) task batching, (2) resource sharing. At a high level, we first solve **P1'**, and then color the graph G based on the solution of **P1'** such that each color induces an IS that can be mapped into a feasible scheduling policy.

Let $\hat{N} \triangleq \pi\delta^{1+\epsilon}$. To make our analysis concise, we assume that $\hat{N}$ is an integer, otherwise we need to round it to the nearest integer. Let $\mathcal{C}$ denote an ordered set (or a vector) of $(2\hat{N}-1)$ colors. For each slot t, each machine k and each group b, let $\mathcal{C}_{b,t,k}$ denote the set of colors already used to color the vertices $v \in \mathcal{V}$ with $t_v \leq t \leq t_v + l_v - 1$ corresponding to group b and machine k.

We solve **P1'** and denote the solution by $\{y_v^*\}_{v\in\mathcal{V}}$, where y_v^* is the value of y_v in the LP relaxation. Clearly, we have $0 \leq y_v^* \leq 1, \forall v \in \mathcal{V}$. Then, we color all vertices in $\mathcal{V}$ with as few colors as possible such that (1) each vertex $v \in \mathcal{V}$ receives $\lfloor \hat{N} y_v^* \rfloor$ colors, (2) any color used to any vertex v is not used to color any neighbor of v. Technically, we first sort the vertices in $\mathcal{V}$ non-decreasingly by their left boundaries, i.e., their starting time, with ties broken randomly. Step 2 is then executed in iterations. In each iteration, we color a vertex in $\mathcal{V}$. For each vertex $v \in \mathcal{V}$ from left to right with respect to the above ordering. Let b denote the group to which the task covered by v belongs, and let k denote the machine to which vertex v corresponds. Let $\hat{\mathcal{C}}$ denote the set of colors in $\mathcal{C}_{b,t_v,k}$ not yet used by any neighbor of v. If there are at least $\lfloor \hat{N} y_v^* \rfloor$ colors in $\hat{\mathcal{C}}$, i.e., $|\hat{\mathcal{C}}| \geq \lfloor \hat{N} y_v^* \rfloor$, we color v using the first $\lfloor \hat{N} y_v^* \rfloor$ colors in $\hat{\mathcal{C}}$; otherwise, we color v using colors in $\hat{\mathcal{C}}$ and the first $\lfloor \hat{N} y_v^* \rfloor - |\hat{\mathcal{C}}|$ colors in $\mathcal{C}$ not yet used by any neighbor of v.

The pseudo-code of our algorithm is given in Algorithm 1. Our algorithm outputs the set of vertices sharing one common color with maximum weight. As each vertex corresponds to a task, the final scheduling policy is to execute the tasks corresponding to these vertices. In the algorithm, we use the functions, which are briefly described

as follows. As the functions are graph algorithms that can be coded straightforwardly, the detailed implementation is thus omitted in the pseudo-code.

- **SameGroupColors**$(b, t, k, \mathcal{C})$ returns the set of colors in $\mathcal{C}$ already used to color the vertices u with $t_u \leq t \leq t_u +_u -1$ corresponding to group b and machine k.
- **AvailableSameGroupColors**$(v, \mathcal{C}^*)$ returns the set of colors in $\mathcal{C}^*$ not yet used by any neighbor of v if such colors exist, and $\emptyset$ otherwise.
- **AvailableColors**$(v, n, \mathcal{C}^*)$ returns the first n colors in $\mathcal{C}^*$ not yet used by any neighbor of v.

Algorithm 1 Batching task scheduling: executed by the scheduler

Input: task graph $G = (\mathcal{V}, \mathcal{E})$, solution of **P1'** $\{y_v^*\}_{v \in \mathcal{V}}$, a vector $\mathcal{C}$ of $(2\hat{N} - 1)$ colors
Output: set of vertices in $\hat{\mathcal{V}}$ ▷ The corresponding scheduling is to execute the tasks corresponding to the vertices in $\hat{\mathcal{V}}$
sort the vertices in $\mathcal{V}$ non-decreasingly by their start-time, with ties broken randomly
for each $v \in \mathcal{V}$ with $\lfloor y_v^* \hat{N} \rfloor \geq 1$ from left to right with respect to the above ordering **do**
    let b denote the group to which the task corresponding to v belongs
    let k denote the machine to which vertex v corresponds
    $\mathcal{C}_{b,t_v,k} \leftarrow$ **SameGroupColors**$(b, t_v, k, \mathcal{C})$
    $\hat{\mathcal{C}} \leftarrow$ **AvailableSameGroupColors**$(v, \mathcal{C}_{b,t_v,k})$
    if $|\hat{\mathcal{C}}| \geq \lfloor y_v^* \hat{N} \rfloor$ **then** ▷ Properly color v
        color v using the colors **AvailableColors**$(v, \lfloor y_v^* \hat{N} \rfloor, \hat{\mathcal{C}})$
    else
        color v using the colors in $\hat{\mathcal{C}}$ and the colors **AvailableColors**$(v, \lfloor y_v^* \hat{N} \rfloor - |\hat{\mathcal{C}}|, \mathcal{C})$
    end if
end for
Return set of vertices in $\mathcal{V}$ sharing one common color with maximum weight

3.3.5 Approximation Ratio Analysis

This subsection is dedicated to the theoretical analysis of our approximation algorithm. We first prove that each vertex $v \in \mathcal{V}$ with $\lfloor y_v^* \hat{N} \rfloor \geq 1$ is colored by $\lfloor y_v^* \hat{N} \rfloor$ colors in Lemma 3.2, based on which we further establish the approximation ratio of Algorithm 1 in Theorem 3.1.

Lemma 3.2. *Each vertex $v \in \mathcal{V}$ with $\lfloor y_v^* \hat{N} \rfloor \geq 1$ is colored by $\lfloor y_v^* \hat{N} \rfloor$ colors by Algorithm 1.*

Proof Sketch. For each vertex $v \in \mathcal{V}$ with $\lfloor y_v^* \hat{N} \rfloor \geq 1$, we prove that there are at least $\lfloor y_v^* \hat{N} \rfloor$ available colors in $\mathcal{C}$ that can be used by Algorithm 1 to color v. □

Theorem 3.1. *Algorithm 1 outputs a $\frac{1}{2}\left(1 - \frac{1}{\delta^\epsilon}\right)$-optimal feasible scheduling policy.*

Proof Sketch. We establish the relationship between the IS output by Algorithm 1 and the utility of an optimal scheduling policy, which allows us to further bound the approximation ratio. □

Asymptotically, by choosing a large δ, the approximation ratio of our algorithm approaches 2. Note that the best approximation ratio in the literature is 4 [6].

We conclude this section by analyzing the complexity of Algorithm 1. Since the number of vertices in $\mathcal{V}$ is upper-bounded by $\pi\delta$, the complexity of the sorting process is upper-bounded by $O(\pi\delta \log(\pi\delta))$. The number of colors received by each vertex is $O(\hat{N})$, and thus the coloring of vertices in $\mathcal{V}$ can be done in $O(\pi\delta\hat{N})$ time. It then follows from $\hat{N} = \pi\delta^{1+\epsilon}$ that the complexity of Algorithm 1 is $O(\pi^2\delta^{2+\epsilon})$. In the case where π is constant, the complexity of Algorithm 1 scales approximately quadratically to the number of requests N.

3.4 The Case of Unbounded Batching

In this section, we describe how our algorithmic framework can be adapted in the unbounded case $\pi_k = N, \forall 1 \leq k \leq K$. Note that the unbounded batching task scheduling problem is still P-complete. Our objective is to adapt our algorithmic framework to produce a 1/2-optimal feasible scheduling policy.

Different from the bounded case, the bottleneck on the number of simultaneously executable tasks on any machine does not exist any more in the unbounded case. As a consequence, the task graph has simpler structure in the unbounded case; the LP relaxation process can then be adapted accordingly. The above adaptation enable us to derive the asymptotic approximation ratio of 2 in unbounded case. In the rest of this section, we briefly explain the main adaptation in the unbounded case.

3.4.1 Task Graph Construction

The task graph $G = (\mathcal{V}, \mathcal{E})$ is constructed below.

Vertices. Consider each task $i \in \mathcal{N}$. For each machine $k \in \mathcal{K}_i$, we create a vertex v for each interval of length $l_{i,k}$ in the time interval $[a_i, d_i]$. We say that v covers i and the machine corresponding to v is k. We denote $\mathcal{V}_i$ the set of vertices covering i, and let $\mathcal{V}_{i,k}$ denote the set of vertices in $\mathcal{V}_i$ corresponding to machine k. For each vertex $v \in \mathcal{V}_{i,k}$, we define a weight w_v and assign w_v to $w_{i,k}$.

Edges. We distinguish the types of edges.

- **Intra-task edges.** For each task i, we construct an edge between each pair of vertices in $\mathcal{V}_i$.
- **Inter-task edges.** Consider each machine k and any pair of tasks $i, j \in \mathcal{N}$ with $b_i \neq b_j$ and $k \in \mathcal{K}_i \cap \mathcal{K}_j$. For any pair of vertices $v \in \mathcal{V}_{i,k}$ and $u \in \mathcal{V}_{j,k}$, we construct an edge between v and u if their corresponding time intervals overlap.

Based on the proof of Lemma 3.1, we can see that there exists a one-to-one mapping between an IS of G and a feasible scheduling policy for unbounded batching without conflict. The unbounded batching scheduling problem is thus transformed into finding a MWIS in G, the formulation of which is the same as in Corollary 3.1.

3.4.2 LP Relaxation

The construction of the LP relaxation in the unbounded case can be significantly simplified. For each slot t, let $\mathcal{V}_{i,k}(t)$ denote the set of vertices $v \in \mathcal{V}_{i,k}$ with $t_v \leq t \leq t_v + l_v - 1$. Consider each machine k. For each slot t and each group b, at any feasible scheduling policy, any task belonging to group b can be served at slot t on machine k if and only if $q_{b,t,k} = 1$. Therefore, for each task $i \in \mathcal{N}$, the sum of y_v's of vertices $v \in \mathcal{V}_{i,k}(t)$ is upper-bounded by $q_{b_i,t,k}$. Mathematically we have the following constraints.

$$\sum_{v \in \mathcal{V}_{i,k}(t)} y_v \leq q_{b_i,t,k}, \quad \forall i \in \mathcal{N},\ 0 \leq t \leq T,\ 1 \leq k \leq K. \tag{3.5}$$

We construct the following LP relaxation of the formulated MWIS problem.

$$\begin{aligned}
&\max \sum_{v \in \mathcal{V}} w_v \cdot x_v \\
&\text{s.t. } (3.2), (3.3), (3.5) \\
&\quad y_v \geq 0, && \forall v \in \mathcal{V}
\end{aligned}$$

$$q_{b,t,k} \geq 0, \qquad \forall 1 \leq b \leq B,\ 0 \leq t \leq T,\ 1 \leq k \leq K.$$

3.4.3 Approximation Algorithm

Armed with the task graph and the LP relaxation, we then set $\hat{N} = \delta^{1+\epsilon}$ in Algorithm 1. We first prove that each vertex $v \in \mathcal{V}$ with $\lfloor y_v^* \hat{N} \rfloor \geq 1$ is colored by $\lfloor \hat{y}_v^* \hat{N} \rfloor$ colors by Algorithm 1 in Lemma 3.3, based on which, we can establish the performance of Algorithm 1 in Theorem 3.2.

Lemma 3.3. *Each vertex $v \in \mathcal{V}$ with $\lfloor y_v^* \hat{N} \rfloor \geq 1$ is colored by $\lfloor \hat{y}_v^* \hat{N} \rfloor$ colors by Algorithm 1.*

Proof. For each vertex $v \in \mathcal{V}$ with $\lfloor y_v^* \hat{N} \rfloor \geq 1$, we prove that there are at least $\lfloor \hat{y}_v^* \hat{N} \rfloor$ available colors in $\mathcal{C}$ that can be used by Algorithm 1 to color the vertex. □

We can then establish the performance bound of Algorithm 1 in the unbounded case. The proof follows the same way as that of Theorem 3.1.

Theorem 3.2. *Algorithm 1 outputs an asymptotically 1/2-optimal feasible scheduling policy for the unbounded batching scheduling problem.*

In the single-machine case, i.e., $K = 1$, when the task slackness is constrained such that $d_i - a_i + 1 < 2l_{i,1}$ for each task i, Algorithm 1 is a PTAS. This result implies that our algorithm is a PTAS for the canonical broadcast scheduling problem in this case.

Theorem 3.3. *When $K = 1$, if $d_i - a_i + 1 < 2l_{i,1}$, $\forall i \in \mathcal{N}$, Algorithm 1 is a PTAS.*

Proof Sketch. We first prove that Algorithm 1 uses at most $\hat{N}$ colors to color all vertices in G. Then, we prove that Algorithm 1 outputs a $(1 - 1/\delta^{\epsilon})$-optimal feasible policy. □

3.5 Applying Our Algorithmic Framework to Solve Generalized Proportional Broadcast Scheduling Problem

In this section, we demonstrate how the developed algorithmic framework can be adapted to solve other scheduling problems. Technically, we consider the generalization of the *proportional broadcast scheduling problem* formulated in [18] as described below.

3.5.1 Problem Statement

In the original proportional broadcast scheduling problem, we are given a set $\mathcal{N}$ of N tasks. Each task i is submitted at time a_i, requires l_i slots (the length of task i) to complete. The deadline of task i is d_i. Different from the standard broadcast scheduling problem, each task does not need to be executed completely and the execution can be interrupted and can be resumed at the last execution point. A reward w_i is generated to the system if task i is executed l_i slots during the time interval $[a_i, d_i]$. In the case where task i is not completely executed, the system still gets a reward that is proportional to the execution time γ_i, i.e., $w_i(\gamma_i) = \frac{\gamma_i}{l_i} w_i$. Tasks are divided into B non-overlapping groups. Tasks in the same group can be executed simultaneously. The goal is to maximize the total reward of the executed tasks. In the original formulation of Gandhi *et al.* [18], the lengths of all tasks are identical, i.e., $l_i = l, \forall i \in \mathcal{N}$. A *randomized approximation algorithm* via dependent rounding scheme is proposed in [18] achieving an expected approximation ratio $\frac{4l}{4l-1}$. Table 3.2 lists the notations used in Section 3.5.

In this section, we consider a generic formulation with *heterogeneous task lengths*. We develop a *deterministic* approximation algorithm by integrating the idea developed in [18] into our algorithmic framework. Our algorithm is the first deterministic algorithm for the proportional broadcast scheduling problem and achieves an asymptotic approximation ratio of $2(l_{min}+1)/l_{min}$, where $l_{min} \triangleq \min_{i \in \mathcal{N}} l_i$.

Our generalized proportional broadcast scheduling problem, denoted by **P2**, can be formulated below [18].

$$
\begin{aligned}
\textbf{P2}: \quad & \max \sum_{i \in \mathcal{N}} w_i \frac{\gamma_i}{l_i} \\
& \text{s.t.} \sum_{t=a_i}^{d_i} q_{b_i,t} \geq \gamma_i, && \forall i \in \mathcal{N} && (3.6) \\
& \sum_{1 \leq b \leq B} q_{b,t} \leq 1, && \forall 0 \leq t \leq T && (3.7) \\
& \gamma_i \in \{0, 1, \cdots, l_i\}, && \forall i \in \mathcal{N} \\
& q_{b,t} \in \{0, 1\}, && \forall 1 \leq b \leq B,\ 0 \leq t \leq T
\end{aligned}
$$

where $q_{b,t}$ is a binary variable indicating whether at least one task of group b is executed or not at slot t. The first constraint ensures that whenever a task i is executed γ_i slots, there are at least γ_i time slots to execute the tasks of group b_i during the interval $[a_i, d_i]$.

Table 3.2: Additional notations used in Section 3.5

γ_i	number of slots executing task i
l_i	length of task i
w_i	reward of task i if it is executed completed
$q_{b,t}$	binary variable indicating whether at least one task of group b is executed at slot t
$q^*_{b,t}$	value of $q_{b,t}$ in LP relaxation
$\mathcal{T}_b$	set of slots $\{t_1, t_2, \cdots, t_m\}$ such that $t_i < t_{i+1}, q^*_{b,t_i} > 0$
m_b	$\left\lceil \sum_{t \in \mathcal{T}_b} q^*_{b,t} \right\rceil$
$\mathcal{W}^b_k$	window consisting of slot t with $z^b_{t,k} > 0$
G_0	bipartite graph $G_0 = (\mathcal{U}, \mathcal{V}, \mathcal{E}_0)$
G_1	bipartite graph $G_1 = (\mathcal{U}, \mathcal{V}, \mathcal{E}_1)$
G_2	bipartite graph $G_2 = (\mathcal{U}, \mathcal{V}_2, \mathcal{E}_2)$
$\mathcal{E}_0$	set of edges in G_0
$\mathcal{E}_1$	set of edges in G_1
$\mathcal{E}_2$	set of edges in G_2
$\mathcal{U}$	set of vertices, each of which corresponds to a slot
u_t	vertex in $\mathcal{U}$ corresponding to slot t
$\mathcal{V}$	set of vertices $\{v^b_1, v^b_2, \cdots, v^b_{m_b}\}_{1 \le b \le B}$
v^b_k	vertex in $\mathcal{V}$ corresponding to group b and window $\mathcal{W}^b_k$
$\mathcal{V}_2$	set of vertices in G_2
$v_{i,k}$	vertex in $\mathcal{V}_2$ corresponding to task i and window $\mathcal{W}^{b_i}_k$
$z^b_{t,k}$	weight of the edge $u_t v^b_k$ in graph G_0
λ_i	$d_i - a_i + 1$
λ	$\sum_{i \in \mathcal{N}} \lambda_i$
$\hat{N}$	scale factor. $\hat{N} = \lambda^{1+\epsilon}$ for the proportional broadcast scheduling problem
$\hat{\mathcal{E}}$	set of edges output by Algorithm 2
$\hat{\mathcal{E}}_i$	set of edges in $\hat{\mathcal{E}}$ covering task i
l_{min}	$\min_{i \in \mathcal{N}} l_i$
γ^*_i	value of γ_i in LP relaxation
w_e	weight of edge e in $\mathcal{E}_2$

The second constraint ensures that any pair of tasks from different groups cannot be executed simultaneously.

The feasible scheduling policy for the generalized proportional broadcast scheduling is defined as below. Our objective in **P2** is to find a feasible scheduling policy maximizing the system reward.

Definition 3.2 (Feasible Scheduling Policy for Generalized Proportional Broadcast Scheduling)**.** *We call a scheduling is feasible for generalized proportional broadcast scheduling if*

- *each task $i \in \mathcal{N}$ is executed at most l_i slots;*
- *executions of any pair of tasks belonging to different groups do not overlap in time.*

3.5.2 Deterministic Approximation Scheduling Algorithm

We first solve the LP relaxation of **P2**. To this end, we let all variables γ_i and $q_{b,t}$ to be reals. Specifically, we replace the constraints $\gamma_i \in \{0, 1, \cdots l_i\}, \forall i \in \mathcal{N}$ with $0 \leq \gamma_i \leq l_i, \forall i \in \mathcal{N}$ and $q_{b,t} \in \{0, 1\}$ by $q_{b,t} \geq 0$. There is no need to explicitly add the constraint $q_{b,t} \leq 1$ since it is implied by (3.7).

The core part of our algorithm is to round the solution of the LP relaxation to a feasible scheduling policy. Gandhi *et al.* developed a randomized rounding algorithm in [18]. However, their algorithm assumes identical l_i and only gives average performance guarantee. In order to design a deterministic scheduling algorithm, we apply our algorithmic framework, more precisely, the rounding and the related coloring technique we develop in previous sections. Our algorithm also integrates the idea used in [18] adapted in our context.

At a high level, our main idea is to construct a series of bipartite graphs, in which each edge captures the relationship between a slot and tasks of a same group. Each graph is constructed based on its predecessor to gradually arrive at a feasible scheduling policy. Instead of coloring vertices, our algorithm now colors the edges in the constructed graphs so that each color induces a matching of the graph or maps to a feasible scheduling policy. Specifically, our algorithm first constructs a bipartite graph G_0 based on the optimum solution of the LP relaxation. Then, we generate an auxiliary bipartite graph G_1 from G_0, and we color the edges in G_1 such that each color induces a matching of the graph G_1. Finally, we construct another edge-colored auxiliary bipartite graph G_2 based on the colored graph G_1 such that each color in G_2 induces a

feasible scheduling policy. The algorithm is composed of five steps, which are exposed sequentially. The first step essentially follows the procedures in [18], which are adapted below in the context of our problem.

Step 1: Construct a bipartite graph $G_0 = (\mathcal{U}, \mathcal{V}, \mathcal{E}_0)$**.** Let $\{\gamma_i^*\}_{i\in\mathcal{N}}$ and $\{q_{b,t}^*\}_{1\le b\le B, 0\le t\le T}$ denote the solution of the LP relaxation of **P2**. G_0 contains two sets of vertices $\mathcal{U}$ and $\mathcal{V}$, constructed as follows.

- Each vertex in $\mathcal{U}$ represents a time slot. Let u_t denote the vertex in $\mathcal{U}$ corresponding to time slot t.

- Each vertex in $\mathcal{V}$ represents a group. For each group b, we seek the time slots, at each of which tasks of group b are executed fractionally in the LP solution, and we denote these time slots by $\mathcal{T}_b = \{t_1, \cdots, t_h, \cdots\}$, where $t_h < t_{h+1}$ and $q_{b,t_h}^* > 0$. We construct $m_b \triangleq \left\lceil \sum_{t=\mathcal{T}_b} q_{b,t}^* \right\rceil$ vertices in $\mathcal{V}$, denoted by $\{v_1^b, v_2^b, \cdots, v_{m_b}^b\}$. Each vertex $v_k^b, 1 \le k \le m_b$, corresponds to a window, denoted by $\mathcal{W}_k^b$, as described later.

- **Edges.** Consider each group b. We group these slots into m_b windows, denoted by $\mathcal{W}_k^b, 1 \le k \le m_b$. To this end, we recursively define non-negative numbers $z_{t_h,k}^b$, $1 \le h \le |\mathcal{T}_b|$ and $1 \le k \le m_b$, as follows.

$$z_{t_h,k}^b \triangleq \begin{cases} A_1 & \text{if } \sum_{s=1}^{h-1} q_{b,t_s}^* < k \text{ and } \sum_{s=1}^{h} q_{b,t_s}^* > k-1 \\ 0 & \text{otherwise} \end{cases},$$

  where $z_{t_h,0}^b = 0$ and

$$A_1 = \min\left\{ q_{b,t_h}^* - z_{t_h,k-1}^b, 1 - \sum_{t'<t_h, t'\in\mathcal{W}_k^b} z_{t',k}^b \right\}.$$

  The time slots t with $z_{t,k}^b > 0$ form the window $\mathcal{W}_k^b$. We can check that $\sum_{t\in\mathcal{W}_k^b} z_{t,k}^b = 1$ for $\forall 1 \le k \le m_b - 1$. In this regard, $z_{t_h,k}^b$ represents the amount of fractional value to execute tasks belonging to group b at slot $t_h \in \mathcal{W}_k^b$. We connect each vertex $v_k^b \in \mathcal{V}$, $1 \le k \le m_b$, to each vertex in $\mathcal{U}$ corresponding to time slots in the window $\mathcal{W}_k^b$. The weight of the edge $u_t v_k^b$ is set to $z_{t,k}^b$.

The construction of G_0 is illustrated in Figure 3.1, in which a subgraph of G_0 related to a single group b is shown.

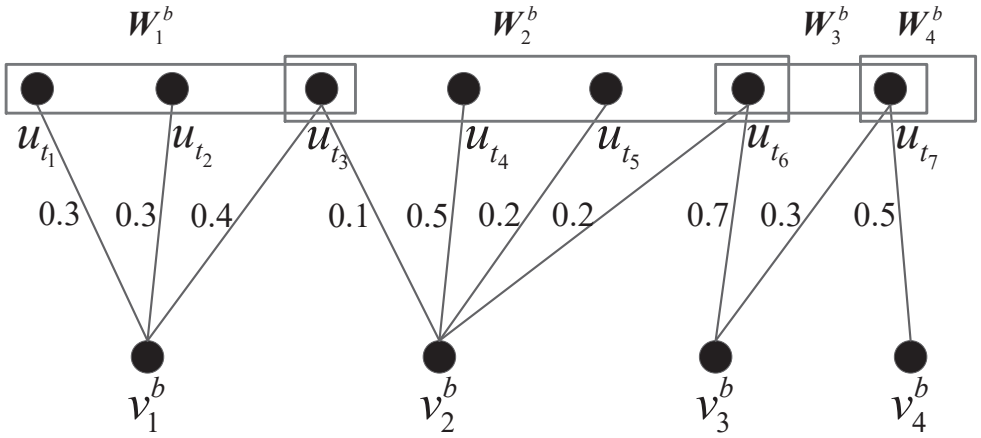


Figure 3.1: Illustration of a subgraph of G_0 related to group b: q^*_{b,t_h}, $1 \leq h \leq 7$, are $0.3, 0.3, 0.5, 0.5, 0.2, 0.9, 0.8$

Step 2: Generate an auxiliary bipartite graph $G_1 = (\mathcal{U}, \mathcal{V}, \mathcal{E}_1)$. We construct G_1 by duplicating the vertices in G_0. Let $\hat{N} \triangleq \lambda^{1+\epsilon}$, where $\lambda \triangleq \sum_{i \in \mathcal{N}} \lambda_i$, and $\lambda_i \triangleq d_i - a_i + 1$. To make our analysis concise, we assume that $\hat{N}$ is an integer, otherwise we need to round it to the nearest integer. Concretely, for each edge $u_t v_k^b \in \mathcal{E}_0$, we construct $\left\lfloor z_{t,k}^b \hat{N} \right\rfloor$ new edges, each of which connects u_t with v_k^b in G_1, and add the constructed edges to $\mathcal{E}_1$. Each constructed edge is a duplicate of the original edge and has the same endpoints. By the definition of $z_{t,k}^b$ in the first step, we have $0 \leq z_{t,k}^b \leq 1$ for any triple (t, k, b). Therefore, each original edge in G_0 is duplicated to at most $\hat{N}$ edges in G_1.

Step 3: Color G_1. We color each edge in G_1 by a color such that no adjacent edges are of the same color. Let $\mathcal{C}$ denote an ordered set of $(2\hat{N} - 1)$ colors. The coloring process runs in iterations. In each iteration, we color an edge using the first color in $\mathcal{C}$ not yet used to color any neighbor of it.

Step 4: Construct another edge-colored auxiliary bipartite graph $G_2 = (\mathcal{U}, \mathcal{V}_2, \mathcal{E}_2)$ based on G_1.

- **Vertices in $\mathcal{V}_2$.** Consider each task $i \in \mathcal{N}$. For each integer $k \in [1, m_{b_i}]$, if the window $\mathcal{W}_k^{b_i}$ includes at least one slot $t \in [a_i, d_i]$ with $z_{t,k}^{b_i} > 0$, i.e., $\mathcal{W}_k^{b_i} \cap [a_i, d_i] \neq \emptyset$, we create a vertex $v_{i,k}$, and add it to $\mathcal{V}_2$.

- **Edges in $\mathcal{E}_2$.** Consider each task i and each integer $k \in [1, m_{b_i}]$. If $\mathcal{W}_k^{b_i} \cap [a_i, d_i] \neq$

$\emptyset$, for each slot $t \in \mathcal{W}_k^{b_i} \cap [a_i, d_i]$, we choose an arbitrary set of

$$\min\left\{\left\lfloor z_{t,k}^{b_i}\hat{N}\right\rfloor, \max\left\{0, l_i\hat{N} - \sum_{k'=1}^{<k}\sum_{t'=a_i}^{t}\left\lfloor z_{t',k'}^{b_i}\hat{N}\right\rfloor - \sum_{t'\in\mathcal{W}_k^{b_i}, t'<t}\left\lfloor z_{t',k}^{b_i}\hat{N}\right\rfloor\right\}\right\}$$

colored edges in $\mathcal{E}_1$ between $u_t \in \mathcal{U}$ and $v_k^{b_i} \in \mathcal{V}$ to connect $v_{i,k} \in \mathcal{V}_2$ with $u_t \in \mathcal{U}$, and add them to $\mathcal{E}_2$. We say that these edges *cover* task i, and assign w_i/l_i as the weight for each of them. Let w_e denote the weight of edge e in $\mathcal{E}_2$.

Step 5: Seek a set of edges, denoted by $\hat{\mathcal{E}}$, of the same color with maximum total weight and map them to a feasible scheduling policy. The final scheduling policy is to execute the tasks at the slots corresponding to the edges in $\hat{\mathcal{E}}$ after a pruning process. To see the necessity of pruning $\hat{\mathcal{E}}$, let $\hat{\mathcal{E}}_i$ denote the edges in $\hat{\mathcal{E}}$ covering task i. We can upper-bound $|\hat{\mathcal{E}}_i|$ by $l_i + 1$. This is because there are at most $l_i\hat{N}$ edges in $\mathcal{E}_2$ covering i; hence the slots corresponding to the $l_i\hat{N}$ edges belong to at most $l_i + 1$ windows $\mathcal{W}_k^{b_i}, k \in [1, m_{b_i}]$ by noticing $\sum_{t\in\mathcal{W}_k^{b_i}} z_{t,k}^{b_i}\hat{N} = \hat{N}, 1 \le k < m_{b_i}$ and $\sum_{t\in\mathcal{W}_{m_{b_i}}^{b_i}} z_{t,m_{b_i}}^{b_i}\hat{N} \le \hat{N}$. However, as each edge in $\hat{\mathcal{E}}$ maps to a slot and each task i is executed at most l_i slots, we need to remove an edge from $\hat{\mathcal{E}}_i$ if $|\hat{\mathcal{E}}_i| = l_i + 1$. We thus run a pruning procedure by removing the edge corresponding to the latest slot. We will prove in Lemma 3.5 that the edges in $\hat{\mathcal{E}}$ after pruning map to a feasible policy.

The pseudo-code of our scheduling algorithm is given in Algorithm 2. The algorithm outputs a set of edges $\hat{\mathcal{E}}$. For each edge in $\hat{\mathcal{E}}$, one of its endpoints corresponds to a time slot, and the other point corresponds to a task. Thus, the final scheduling is to execute the tasks at the time slots corresponding to the set of edges output by Algorithm 2. In the algorithm, the following functions are invoked. Their implementation is rather straightforward and omitted here.

- **Color**$(e, \mathcal{C})$ returns the first color in $\mathcal{C}$ not yet used to color any neighbor edge of e.

- **ExecutedSlots**(b) returns an ordered set of slots $\{t_1, t_2, \cdots, t_h, \cdots\}$, $t_h < t_{h+1}$, at each of which the tasks of group b are executed fractionally in the LP solution, i.e, $q_{b,t_h}^* > 0$.

- **Edges**$(t, u_t, v_k^b, \mathcal{E}_1)$ returns an arbitrary set of

$$\min\left\{\left\lfloor z_{t,k}^{b_i}\hat{N}\right\rfloor, \max\left\{0, l_i\hat{N} - \sum_{k'=1}^{<k}\sum_{t'=a_i}^{t}\left\lfloor z_{t',k'}^{b_i}\hat{N}\right\rfloor - \sum_{t'\in\mathcal{W}_k^{b_i}, t'<t}\left\lfloor z_{t',k}^{b_i}\hat{N}\right\rfloor\right\}\right\}$$

  colored edges in $\mathcal{E}_1$ between $u_t \in \mathcal{U}$ and $v_k^b \in \mathcal{V}$.

- **MaxWeightEdges**$(\mathcal{E}_2)$ returns the set of edges in $\mathcal{E}_2$ of a same color with maximum weight.

- **EdgesCoverTask**$(i, \hat{\mathcal{E}})$ returns the set of edges in $\hat{\mathcal{E}}$ covering task i.

- **LatestEdge**$(\hat{\mathcal{E}}_i)$ returns the edge in $\hat{\mathcal{E}}_i$ corresponding to the latest slot.

Algorithm 2 Proportional broadcast scheduling: executed by the scheduler

Input: solution of LP relaxation of **P2** $\{q_{b,t}^*\}_{1\le b\le B, 0\le t\le T}$, $\hat{N} = \lambda^{1+\epsilon}$, a vector $\mathcal{C}$ of $2\hat{N}-1$ colors
Output: $\hat{\mathcal{E}}$ ▷ The scheduling is to execute the tasks at slots corresponding to the edges in $\hat{\mathcal{E}}$
$\mathcal{U} \leftarrow \emptyset, \mathcal{V} \leftarrow \emptyset, \mathcal{E}_0 \leftarrow \emptyset$ ▷ Construct a bipartite graph $G_0 = (\mathcal{U}, \mathcal{V}, \mathcal{E}_0)$
for each slot t **do**
    create a vertex u_t, and add u_t to $\mathcal{U}$ ▷ $\mathcal{U}$ consists of vertices representing slots
end for
for each group b **do**
    $\mathcal{T}_b \leftarrow$ **ExecutedSlots**(b)
    $m_b \leftarrow \left\lceil \sum_{t\in\mathcal{T}_b} q_{b,t}^* \right\rceil$
    for $k = 1$ to m_b **do**
        create a vertex v_k^b, and add it to $\mathcal{V}$
        $\mathcal{W}_k^b \leftarrow \emptyset$
        for $h = 1$ **to** $|\mathcal{T}_b|$ **do**
            if $\sum_{s=1}^{h-1} q_{b,t_s}^* < k$ and $\sum_{s=1}^{h} q_{b,t_s}^* > k-1$ **then**
                $z_{t_h,k}^b \leftarrow \min\left\{q_{b,t_h}^* - z_{t_h,k-1}^b, 1 - \sum_{t'<t_h, t'\in\mathcal{W}_k^b} z_{t',k}^b\right\}$
            else
                $z_{t_h,k}^b \leftarrow 0$
            end if
            if $z_{t_h,k}^b > 0$ **then**
                create an edge e between u_{t_h} and v_k^b
                add e to $\mathcal{E}_0$
                add t_h to $\mathcal{W}_k^b$
            end if
        end for
    end for
end for

$\mathcal{E}_1 \leftarrow \emptyset$ ▷ Construct an auxiliary bipartite graph $G_1 = (\mathcal{U}, \mathcal{V}, \mathcal{E}_1)$
for each edge $u_t v_k^b \in \mathcal{E}$ **do**
    construct $\left\lfloor \hat{N} z_{t,k}^b \right\rfloor$ edges between $u_t \in \mathcal{U}$ and $v_k^b \in \mathcal{V}$, and add them to $\mathcal{E}_1$
end for
for each edge $e \in \mathcal{E}_1$ **do** ▷ Properly color the constructed edges
    color e using the color **Color**$(e, \mathcal{C})$
end for
$\mathcal{V}_2 \leftarrow \emptyset, \mathcal{E}_2 \leftarrow \emptyset$ ▷ Construct another auxiliary bipartite graph $G_2 = (\mathcal{U}, \mathcal{V}_2, \mathcal{E}_2)$
for each task $i \in \mathcal{N}$ **do**
    for $k = 1$ **to** m_{b_i} **do**
        if $\mathcal{W}_k^{b_i} \cap [a_i, d_i] \neq \emptyset$ **then**
            create a vertex $v_{i,k}$, and add $v_{i,k}$ to $\mathcal{V}_2$
            for $t = \min\{\mathcal{W}_k^{b_i} \cap [a_i, d_i]\}$ **to** $\max\{\mathcal{W}_k^{b_i} \cap [a_i, d_i]\}$ **do**
                $\bar{\mathcal{E}} \leftarrow$ **Edges**$(t, u_t, v_k^{b_i}, \mathcal{E}_1)$
                use the edges in $\bar{\mathcal{E}}$ connect $v_{i,k} \in \mathcal{V}_2$ with $u_t \in \mathcal{U}$
                assign w_i/l_i to the weight for each of the edges in $\bar{\mathcal{E}}$
                add $\bar{\mathcal{E}}$ to $\mathcal{E}_2$
            end for
        end if
    end for
end for
$\hat{\mathcal{E}} \leftarrow$ **MaxWeightEdges**$(\mathcal{E}_2)$
for each task $i \in \mathcal{N}$ **do** ▷ Prune the edges in $\hat{\mathcal{E}}$
    $\hat{\mathcal{E}}_i \leftarrow$ **EdgesCoverTask**$(i, \hat{\mathcal{E}})$
    if $|\hat{\mathcal{E}}_i| == l_i + 1$ **then**
        remove the edge **LatestEdge**$(\hat{\mathcal{E}}_i)$ from $\hat{\mathcal{E}}$
    end if
end for
return $\hat{\mathcal{E}}$

3.5.3 Approximation Ratio Analysis

In this subsection we derive the theoretical performance guarantee of Algorithm 2. We first prove that all edges in $\mathcal{E}_1$ are colored in Lemma 3.4, and Algorithm 2 outputs the set of edges mapping to a feasible scheduling policy in Lemma 3.5. We are then able to establish the approximation factor of Algorithm 2 in Theorem 3.4. We conclude this subsection by giving the complexity of Algorithm 2.

Lemma 3.4. *All the edges in $\mathcal{E}_1$ are colored by Algorithm 2.*

Proof Sketch. For each edge in $\mathcal{E}_1$, we prove that there is at least one available color in $\mathcal{C}$ that can be used by Algorithm 2 to color the edge. □

Lemma 3.5. *The scheduling policy corresponding to the edges in $\hat{\mathcal{E}}$ output by Algorithm 2 is feasible.*

Proof Sketch. We first prove that any pair of tasks belonging to different groups are not executed simultaneously in the scheduling policy at the same slot. We then prove that there are at most l_i time slots to execute task i in the scheduling policy. The feasibility of the output scheduling policy follows from the above results. □

Theorem 3.4. *Algorithm 2 outputs an asymptotic $\frac{l_{min}}{2(l_{min}+1)}$-optimal feasible scheduling policy.*

Proof Sketch. We first establish the relationship between the total weight of $\hat{\mathcal{E}}$ before pruning, i.e., $\hat{\mathcal{E}}$ at line 48 of Algorithm 2, and the utility of the optimal scheduling policy. We then establish the relationship between the the total weight of $\hat{\mathcal{E}}$ before and after pruning. Combining the above obtained result allows us to establish the $\frac{l_{min}}{2(l_{min}+1)}$-optimality of Algorithm 2. □

To conclude this section, we analyse the complexity of Algorithm 2. Because there are at most λ_i time slots at which each task i is executed fractionally in the LP relaxation, the number of time slots in $\mathcal{T}_b$ is $O(\sum_{i\in\mathcal{N}_b}\lambda_i)$. Based on the construction of G_0, each slot in $\mathcal{T}_b$ refers to at most two edges in $\mathcal{E}_0$ corresponding to group b, and thus the number of edges in G_0 is $O(\sum_{i\in\mathcal{N}}\lambda_i) = O(\lambda)$. Thus, the construction of G_0 can be done in $O(\lambda)$ time. As each edge $u_t v_k^b \in \mathcal{E}_0$ is duplicated by $\lfloor z_{t,k}^b \hat{N} \rfloor$ edges in $\mathcal{E}_1$, the number of edges in $\mathcal{E}_1$ is $O(\lambda \cdot \hat{N})$, and thus it follows from $|\mathcal{C}| = O(\hat{N})$ that the complexity for coloring of the edges in $\mathcal{E}_1$ is $O(\lambda \cdot \hat{N}^2)$. Since the number of edges for each task i is upper-bounded by $l_i\hat{N}$, the number of edges in $\mathcal{E}_2$ can be upper-bounded by $O(\sum_{i\in\mathcal{N}} l_i\hat{N})$. It then follows from $\hat{N} = \lambda^{1+\epsilon}$ that the complexity of Algorithm 2 is $O(\lambda^{3+2\epsilon})$, asymptotically $O(\lambda^3)$.

3.6 Numerical Analysis

In this section, we conduct numerical analysis to evaluate the performance of the constant-factor scheduling approximation algorithms we develop. In our simulation, we trace the following metric to evaluate the performance of the optimum scheduling policy compared to our algorithms:

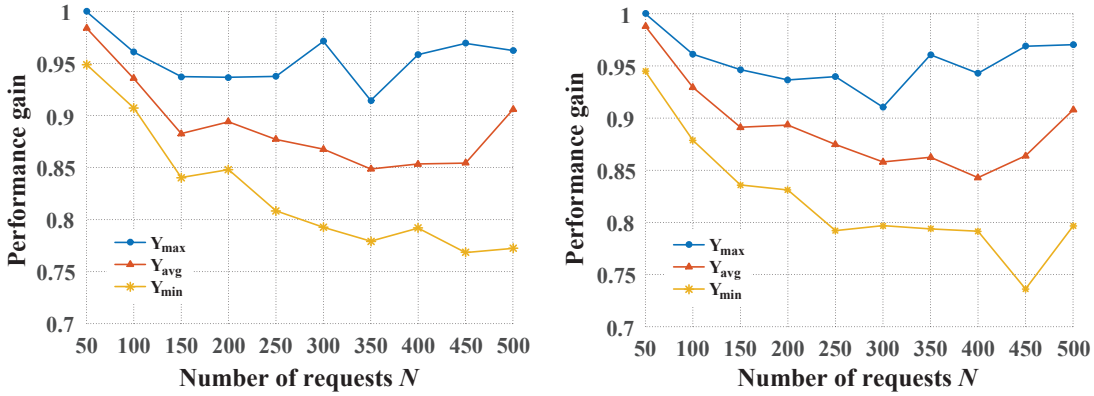
$$\Upsilon \triangleq \frac{\text{system utility of our algorithm}}{\text{system utility under optimal policy}}$$

$$= \frac{\text{total reward of requests served by our algorithm}}{\text{total reward of requests served by optimum policy}} \tag{3.8}$$

Specifically, we trace the maximal, average, and minimal values of Υ in our simulations. We simulate the baseline scenario of batching task scheduling problem including bounded case, unbounded case, and the problem of proportional broadcast scheduling, respectively.

3.6.1 Baseline Scenario of Batching Task Scheduling Problem

In our simulation, the time horizon T is set to 200, and there are 5 machines in the system. We set $l_{max} = 5$ and $l_{min} = 1$. We randomly choose the parameters a_i, d_i, l_i, b_i, w_i such that $d_i - a_i + 1 \geq l_i, \forall i \in \mathcal{N}$, and each request can be executed on any of machines. In the bounded batching case, the maximum batch size that can be supported by each machine is set to 10. In the unbounded batching case, the number of requests in each batch is no limit. We vary the number of requests N in the system from 50 to 500. For each N, we perform 50 simulation runs for each parameter setting. The simulation results of the bounded case and the unbounded case are illustrated in Figures 3.2(a) and 3.2(b). We also analyze our algorithm performance using mean, variance confidence intervals (CI), which are listed in Table 3.3 and 3.4.



(a) Bounded case

(b) Unbounded case

Figure 3.2: Performance gains of Algorithm 1

From the simulation results, we make the following observations.

- Our algorithm achieves at least 76.84% of the optimal utility even in the worst case in the bounded case and at least 73.62% in the unbounded case, which are in accordance to the theoretical results we derive.

Table 3.3: Mean, variance and confidence intervals in bounded case

N	50	100	150	200	250
Mean	0.9843	0.9356	0.8828	0.8940	0.8771
Variance	0.0144	0.0155	0.0230	0.0212	0.0315
CI (95%)	[0.9788,0.9897]	[0.9297,0.9415]	[0.8740,0.8915]	[0.8860,0.9021]	[0.8652,0.8891]
N	300	350	400	450	500
Mean	0.8678	0.8487	0.8535	0.8544	0.9058
Variance	0.0424	0.0292	0.0429	0.0484	0.0449
CI (95%)	[0.8517,0.8839]	[0.8376,0.8598]	[0.8373,0.8698]	[0.8360,0.8728]	[0.8887,0.9229]

Table 3.4: Mean, variance and confidence intervals in unbounded case

N	50	100	150	200	250
Mean	0.9881	0.9295	0.8913	0.8933	0.8747
Variance	0.0149	0.0210	0.0289	0.0276	0.0297
CI (95%)	[0.9824,0.9937]	[0.9215,0.9374]	[0.8802,0.90224]	[0.8828,0.9038]	[0.8634,0.8860]
N	300	350	400	450	500
Mean	0.8642	0.8625	0.8430	0.8639	0.9078
Variance	0.0375	0.0434	0.0326	0.0549	0.0351
CI (95%)	[0.8499,0.8784]	[0.8460,0.8789]	[0.8306,0.8553]	[0.8431,0.8847]	[0.8944,0.9211]

- When the number of requests N increases, the performance gains slightly decrease, but our algorithm always maintains a good results.
- The small variances and confidence intervals indicate the reliability and stabilization of our algorithm.

3.6.2 Proportional Broadcast Scheduling Problem

In our simulation, the time horizon T is set to 200, and we set $l_{max} = 10$ and $l_{min} = 1$. We randomly choose the parameters a_i, d_i, l_i, b_i, w_i such that $d_i - a_i + 1 \geq l_i, \forall i \in \mathcal{N}$. We vary the number of requests N in the system from 50 to 500. For each N, we perform 50 simulation runs for each request parameter. The simulation results of the proportional broadcast scheduling problem are illustrated in Figure 3.3. Mean, variance confidence intervals (CI) are listed in Table 3.5.

From the simulation results, we make the following observations.

- Our algorithm achieves at least 92.08% of the optimal utility, which is in accordance to the theoretical result we derive.
- Wen the number of requests N increases, the performance gains also increase. This

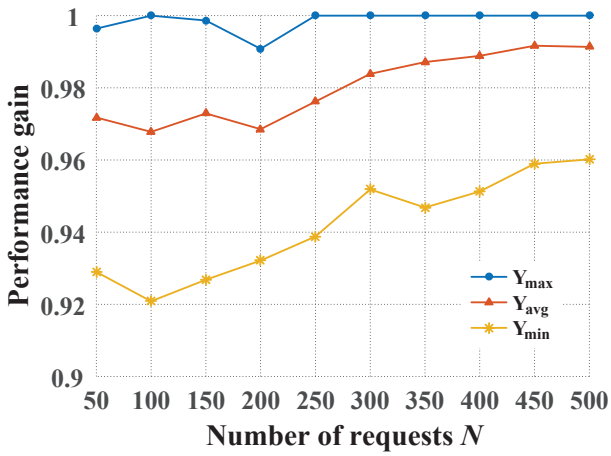


Figure 3.3: Performance gains of Algorithm 2 for the proportional broadcast scheduling problem

is because, when the number of requests is large, there are more opportunities to execute more tasks at each slot due to the nature of the proportional broadcast, which potentially improves the system performance.

Table 3.5: Mean, variance and confidence intervals for proportional broadcast scheduling

N	50	100	150	200	250
Mean	0.9718	0.9678	0.9729	0.9685	0.9762
Variance	0.0140	0.0211	0.0166	0.0154	0.0142
CI (95%)	[0.9677,0.9758]	[0.9617,0.9738]	[0.9682,0.9776]	[0.9641,0.9729]	[0.9721,0.9803]
N	300	350	400	450	500
Mean	0.9839	0.9871	0.9889	0.9917	0.9913
Variance	0.0115	0.0109	0.0108	0.0077	0.0092
CI (95%)	[0.9806,0.9871]	[0.9840,0.9902]	[0.9857,0.9919]	[0.9894,0.9938]	[0.9887,0.9939]

3.7 Conclusion

Motivated by the classic broadcast scheduling problem, we have investigated a class of batching task scheduling problems in its baseline form. We have developed an algorithmic framework achieving 1/2-optimality, outperforming the best known result [6]. The core technicality in our design is an adapted LP relaxation mechanism and a rounding and coloring approach that turns the solution of the LP relaxation to a feasible 1/2-optimal scheduling policy. We have then demonstrated the application of our algorithmic framework to solve the proportional broadcast problem. In this re-

gard, we have developed the first deterministic approximation algorithm outputting a $l_{min}/(2(l_{min}+1))$-optimal scheduling policy. We have complemented our theoretical analysis with numerical simulations that demonstrate the effectiveness of our algorithms. In the following chapters, we investigate two fundamental scheduling problems, which are significant varieties and extensions of the baseline scenario of batching task scheduling problem.

3.8 Appendix

3.8.1 Proof of Lemma 3.1

To prove that each feasible scheduling policy maps to an IS of G, assume by contradiction that there exists a feasible scheduling policy mapping to a subset of vertices $\mathcal{V}$ in G: among these vertices there are two neighboring vertices u and v connected by an edge denoted by e. In the construction of graph G:

- e cannot be an intra-task edge, otherwise u and v correspond to the same task, contradicting with the constraint that each task is executed at most once at any feasible policy.

- e cannot be an inter-task edge characterizing interference conflicts among tasks, otherwise u and v interfere with each other, contradicting with the constraint that the time intervals for executing any pair of interfere tasks on each machine never overlap at any feasible policy.

- e cannot be an inter-task edge modeling bounded batching, otherwise u and v correspond to the same sub-machine and the same machine, and their corresponding time intervals overlap, contradicting with the constraint that at most one task can be executed in each sub-machine of each machine at a same time.

The above analysis demonstrates that e cannot exist, thus proving via contradiction that each feasible scheduling policy maps to an IS of G.

We then prove that each IS of G maps to a feasible scheduling policy.

- The construction of intra-task edges ensures that for each task, at most one vertex covering it is chosen.

- The construction of inter-task edges characterizing conflicts among tasks ensures that the time intervals for executing any pair of interference tasks on each machine (e.g., tasks are not from the same group) never overlap.

- The construction of inter-task edges modeling bounded batching ensures that at most one task can be executed at each sub-machine of each machine at the same time, further ensures that each batch contains at most π_k tasks on each machine k.

It then follows that each IS of G maps to a feasible scheduling policy.

3.8.2 Proof of Lemma 3.2

Consider any vertex $v \in \mathcal{V}$ with $\lfloor y_u^* \hat{N} \rfloor \geq 1$. Denote the task corresponding to v by i and the machine corresponding to v by k. Let $q_{b,t,k}^*$ denote the value of $q_{b,t,k}$ in the LP relaxation. The neighbors of v can be divided into the following two classes.

The first class of neighbors consist of the vertices that are connected with v by inter-task edges. Consider the subgraph, denoted by G_1, of G, in which we remove the intra-task edges from G. Clearly, G_1 includes all first class of v's neighbors. When Algorithm 1 colors v, all vertices $u \in \mathcal{V}$ with $t_u < t_v$ are already colored, while any vertex $u \in \mathcal{V}$ with $t_u > t_v$ is not yet colored. Hence, any color, which cannot be used to color v, already occupied by the first class of neighbors must be already used to color some vertices $u \in \mathcal{V}$ corresponding to machine k with $t_u \leq t_v \leq t_u + l_u - 1$. Consider the time slot t_v and each group b. For each sub-machine r of machine k, it follows from the constraint (3.4) that

$$\sum_{j \in \mathcal{N}_b} \sum_{u \in \mathcal{V}_{j,k,r}(t_v)} \lfloor y_u^* \hat{N} \rfloor \leq \left\lfloor \sum_{u \in \mathcal{V}_{j,k,r}(t_v), j \in \mathcal{N}_b} y_u^* \hat{N} \right\rfloor \leq \left\lfloor q_{b,t_v,k}^* \cdot \hat{N} \right\rfloor.$$

It then holds that the number of colors already used to vertices in $\{\mathcal{V}_{j,k,r}(t_v)\}_{j \in \mathcal{N}_b}$ is upper-bounded by $\left\lfloor q_{b,t_v,k}^* \cdot \hat{N} \right\rfloor$.

Consider each group b. For any pair of sub-machines r_1, r_2, it follows from the construction of edges in G that any vertex in $\{\mathcal{V}_{j,k,r_1}(t_v)\}_{j \in \mathcal{N}_b}$ and any vertex in $\{\mathcal{V}_{j,k,r_2}(t_v)\}_{j \in \mathcal{N}_b}$ are not adjacent to each other in G_1. Therefore, Algorithm 1 uses at most $\left\lfloor q_{b,t_v,k}^* \cdot \hat{N} \right\rfloor$ colors to color all vertices u in G_1 with $t_u \leq t_v \leq t_u + l_u - 1$ corresponding to machine k and group b.

It follows from the constraint (3.3) that Algorithm 1 uses at most $\hat{N}$ colors to color the vertices u in G_1 with $t_u \leq t_v \leq t_u + l_u - 1$. Mathematically,

$$\sum_{1 \leq b \leq B} \left\lfloor q^*_{b,t_v,k} \cdot \hat{N} \right\rfloor \leq \sum_{1 \leq b \leq B} q^*_{b,t_v,k} \cdot \hat{N} \leq \hat{N}. \tag{3.9}$$

Since the above analysis includes vertex v, we can upper-bound the number colors already used to color the neighbors of v in G_1 by $\hat{N} - \lfloor y^*_v \hat{N} \rfloor$.

The second class of neighbors consist of the vertices that are connected with v by intra-task edges. Consider the subgraph, denoted by G_2, of G, in which we remove the inter-task edges from G. For each task j, it follows from the constraint (3.2) that there are at most $\hat{N}$ vertices in G_2 covering task j. Mathematically,

$$\sum_{u \in \mathcal{V}_j} \left\lfloor y^*_u \cdot \hat{N} \right\rfloor \leq \sum_{u \in \mathcal{V}_j} y^*_u \cdot \hat{N} \leq \hat{N}. \tag{3.10}$$

For any pair of tasks j_1, j_2, there is no inter-task edge between any vertex in $\mathcal{V}_{j_1}$ and any vertex in $\mathcal{V}_{j_2}$. Therefore, Algorithm 1 uses at most $\hat{N}$ colors to color G_2, and the number of colors already used to color the second class of neighbors is upper-bounded by $\hat{N} - \lfloor y^*_v \hat{N} \rfloor$.

By combining the above analysis, when Algorithm 1 colors v, there are at most $2(\hat{N} - \lfloor y^*_v \hat{N} \rfloor)$ colors already used to color the neighbors of v. As $|\mathcal{C}| = 2\hat{N} - 1$ and $\lfloor y^*_u \hat{N} \rfloor \geq 1$, there are at least $\lfloor y^*_u \hat{N} \rfloor$ available colors that can be used by Algorithm 1 to color v.

3.8.3 Proof of Theorem 3.1

It is straightforward to observe that each color induces an IS of G. Denote the utility of an optimal scheduling policy by OPT. Let $\mathcal{I}^*$ denote the IS output by Algorithm 1. As each vertex $v \in \mathcal{V}$ is replaced by $\lfloor y^*_v \hat{N} \rfloor$ vertices, we have

$$\begin{aligned}
\sum_{v \in \mathcal{I}^*} w_v &\geq \frac{\hat{N}}{|\mathcal{C}|} \left(\sum_{v \in \mathcal{V}} w_v y^*_v - \sum_{v \in \mathcal{V}} \frac{w_v}{\hat{N}} \right) \\
&\geq \frac{\hat{N}}{|\mathcal{C}|} \left(OPT - \frac{1}{\delta^\epsilon} \max_{v \in \mathcal{V}} w_v \right) \\
&= \frac{\hat{N}}{2\hat{N} - 1} \left(OPT - \frac{1}{\delta^\epsilon} \max_{v \in \mathcal{V}} w_v \right).
\end{aligned}$$

As it holds trivially that $OPT \geq \max_{v \in \mathcal{V}} w_v$. We then have

$$\sum_{v \in \mathcal{I}^*} w_v \geq \frac{\hat{N}}{2\hat{N}-1}\left(1-\frac{1}{\delta^\epsilon}\right) OPT > \frac{1}{2}\left(1-\frac{1}{\delta^\epsilon}\right) OPT.$$

The theorem is thus proved.

3.8.4 Proof of Lemma 3.3

Consider any vertex $v \in \mathcal{V}$ with $\lfloor \hat{y}_v^* \hat{N} \rfloor \geq 1$. Denote the task corresponding to v by i and the machine corresponding to v by k. The neighbors of v can be divided into the following two classes.

The first class of neighbors consist of the vertices that are connected with v by inter-task edges. Consider the subgraph, denoted by G_1, of G, in which we remove the intra-task edges from G. Any color, which cannot be used to color v, already occupied by the first class of neighbors must be already used to color some vertices $u \in \mathcal{V}$ corresponding to machine k with $t_u \leq t_v \leq t_u + l_u - 1$. Consider the time slot t_v and the machine k. For each task $j \in \mathcal{N}$, it follows from the constraint (3.5) that

$$\sum_{u \in \mathcal{V}_{j,k}(t_v)} \lfloor y_u^* \hat{N} \rfloor \leq \left\lfloor \sum_{u \in \mathcal{V}_{j,k}(t_v)} y_u^* \hat{N} \right\rfloor \leq \left\lfloor q_{b_j,t_v,k}^* \cdot \hat{N} \right\rfloor. \tag{3.11}$$

It follows from the above inequality that the number of colors already used to color vertices in $\mathcal{V}_{j,k}(t_v)$ is upper-bounded by $\left\lfloor q_{b_j,t_v,k}^* \cdot \hat{N} \right\rfloor$.

Consider each group b. For any pair of tasks $j_1, j_2 \in \mathcal{N}_b$, any vertex in $\mathcal{V}_{j_1,k}(t_v)$ and any vertex in $\mathcal{V}_{j_2,k}(t_v)$ are not adjacent to each other in G_1. Therefore, Algorithm 1 uses at most $\lfloor q_{b,t_v,k}^* \cdot \hat{N} \rfloor$ colors to color all vertices u in G_1 with $t_u \leq t_v \leq t_u + l_u - 1$ corresponding to machine k and group b.

It follows from (3.9) that Algorithm 1 uses at most $\hat{N}$ colors to color the vertices u in G_1 with $t_u \leq t_v \leq t_u + l_u - 1$. Then, we can upper-bound the number colors already used to color the neighbors of v in G_1 by $\hat{N} - \lfloor \hat{y}_v^* \hat{N} \rfloor$.

The second class of neighbors consist of the vertices that are connected with v by intra-task edges. It follows from the proof of Lemma 3.2 that the number of colors already used to color the second class of neighbors is upper-bounded by $\hat{N} - \lfloor \hat{y}_v^* \hat{N} \rfloor$.

By combining the above analysis, when Algorithm 1 colors v, there are at most

$2(\hat{N} - \lfloor \hat{y}_v^* \hat{N} \rfloor)$ colors already used to color the neighbors of v. As $|\mathcal{C}| = 2\hat{N} - 1$ and $\lfloor \hat{y}_v^* \hat{N} \rfloor \geq 1$, there are at least $\lfloor \hat{y}_v^* \hat{N} \rfloor$ available colors that can be used by Algorithm 1 to color v.

3.8.5 Proof of Theorem 3.3

To prove the theorem, we first prove that Algorithm 1 uses at most $\hat{N}$ colors to color all vertices in G in the following lemma.

Lemma 3.6. *Algorithm 1 uses at most $\hat{N}$ colors to color all vertices in G.*

Proof. Consider each vertex $v \in \mathcal{V}$. Denote the task corresponding to v by i. As $d_j - a_j + 1 < 2l_{j,1}, \forall j \in \mathcal{N}$, any already colored vertex $u \in \mathcal{V}_i$ satisfies $t_u \leq t_v \leq t_u + l_u - 1$. Hence, any color, which cannot be used to color v, must be already used to color some vertices $u \in \mathcal{V}$ with $t_u \leq t_v \leq t_u + l_u - 1$. For the time slot t_v, each task $j \in \mathcal{N}$ and the machine $k = 1$, the number of colors already used to color the vertices in $\mathcal{V}_{j,1}(t_v)$ is upper-bounded by $\left\lfloor q^*_{b_j,t_v,1} \cdot \hat{N} \right\rfloor$ because of (3.11).

Consider each group b. For any pair of tasks $j_1, j_2 \in \mathcal{N}_b$, any vertex in $\mathcal{V}_{j_1,1}(t_v)$ and any vertex in $\mathcal{V}_{j_2,1}(t_v)$ are not adjacent to each other in G. Therefore, Algorithm 1 uses at most $\left\lfloor q^*_{b,t_v,1} \cdot \hat{N} \right\rfloor$ colors to color all vertices $u \in \mathcal{V}$ with $t_u \leq t_v \leq t_u + l_u - 1$ corresponding to group b. It follows from the (3.9) that Algorithm 1 uses at most $\hat{N}$ colors to color the vertices $u \in \mathcal{V}$ with $t_u \leq t_v \leq t_u + l_u - 1$ including vertex v. The lemma is thus proved. □

Let $\mathcal{I}^*$ denote the IS output by Algorithm 1. As each vertex $v \in \mathcal{V}$ is replaced by $\lfloor y_v^* \hat{N} \rfloor$ vertices, we have

$$\sum_{v \in \mathcal{I}^*} w_v \geq \frac{\hat{N}}{\hat{N}} \left(\sum_{v \in \mathcal{V}} w_v y_v^* - \sum_{v \in \mathcal{V}} \frac{w_v}{\hat{N}} \right) \geq \left(OPT - \frac{1}{\delta^\epsilon} \max_{v \in \mathcal{V}} w_v \right).$$

As it holds trivially that $OPT \geq \max_{v \in \mathcal{V}} w_v$. Because Algorithm 1 uses at most $\hat{N}$ colors to color all vertices in $\mathcal{V}'$, we then have

$$\sum_{v \in \mathcal{I}^*} w_v \geq \left(1 - \frac{1}{\delta^\epsilon} \right) OPT.$$

The theorem is thus proved.

3.8.6 Proof of Lemma 3.4

Consider each edge $u_t v_t^b \in \mathcal{E}_1$. The neighbors of $u_t v_t^b$ can be divided into two classes based on the construction of the bipartite graph G_1.

- **The first class of neighbors consist of the edges that are adjacent to $u_t v_k^b$ because of vertex u_t.** For the slot t, it follows from the constraint (3.7) that

$$\begin{aligned}\sum_{1\le b\le B}\sum_{1\le k\le m_b}\left\lfloor z_{t,k}^b \hat{N}\right\rfloor - 1 &\le \sum_{1\le b\le B}\sum_{1\le k\le m_b} z_{t,k}^b \hat{N} - 1 \\ &\le \sum_{1\le b\le B} q_{b,t}^* \cdot \hat{N} - 1 \\ &\le \hat{N} - 1\end{aligned}$$

  Thus, the number of the first class of neighbors is upper-bounded by $\hat{N}-1$. Therefore, when Algorithm 2 colors the edge $u_t v_t^b$, the number of colors that have been used to color the first class of neighbors is upper-bounded by $\hat{N}-1$.

- **The second class of neighbors consist of the edges that are adjacent to $u_t v_k^b$ because of vertex v_k^b.** The window corresponding to vertex v_k^b is $\mathcal{W}_k^b$. It follows from the definition of the windows that $\sum_{t\in\mathcal{W}_k^b} z_{t,k}^b \le 1, \forall b, k$. Therefore, the total number of neighbors in $\mathcal{E}_1$ because of v_k^b is at most $\hat{N}-1$. Mathematically

$$\sum_{0\le t\le T}\lfloor z_{t,k}^b \hat{N}\rfloor - 1 \le \sum_{0\le t\le T} z_{t,k}^b \hat{N} - 1 \le \hat{N} - 1.$$

  Therefore, the number of colors used to color the second class of neighbors is upper-bounded by $\hat{N}-1$.

By combining the above analysis, when Algorithm 2 colors the edge $u_t v_t^b$, the number of colors that have been used to its neighbors is upper-bounded by $2(\hat{N}-1)$. As $|\mathcal{C}| = 2\hat{N}-1$, there is at least one available color that can be used by Algorithm 2 to color the edge $u_t v_t^b$.

3.8.7 Proof of Lemma 3.5

To prove the lemma, we first prove that any pair of tasks belonging to different groups are not executed simultaneously by the scheduling policy at a same slot. For each

slot t, any pair of edges, whose endpoints in $\mathcal{U}$ are u_t, use different colors. Therefore, for any pair of edges in $\hat{\mathcal{E}}$ corresponding to slot t, the tasks covered by them must belong to the same group, otherwise, the edges receive the different colors and cannot both belong to $\hat{\mathcal{E}}$. Therefore, the tasks are executed by the scheduling policy at the same slot must belong to the same group.

Then, it follows from Step 5 that there are at most l_i edges in $\hat{\mathcal{E}}$ covering task i. Therefore, there are at most l_i slots to execute task i in the final scheduling.

By combining the above analysis, we can derive that the scheduling policy is feasible.

3.8.8 Proof of Theorem 3.4

For each task i, it follows from the constraint (3.6) and the construction of G_0 that

$$\sum_{t=a_i}^{d_i} q^*_{b_i,t} \cdot \hat{N} = \sum_{1\le k\le m_{b_i}} \sum_{t\in\mathcal{W}_k^{b_i},[a_i,d_i]\cap\mathcal{W}_k^{b_i}\neq\emptyset} z^{b_i}_{t,k} \cdot \hat{N} \ge \gamma^*_i \cdot \hat{N}. \tag{3.12}$$

Because each edge $u_t v^b_k \in \mathcal{E}_0$ is duplicated by $\lfloor z^b_{t,k}\hat{N}\rfloor$ edges in the construction of G_1, it holds that

$$z^b_{t,k}\hat{N} - 1 \le \lfloor z^b_{t,k}\hat{N}\rfloor \le z^b_{t,k}\hat{N} \le \lfloor z^b_{t,k}\hat{N}\rfloor + 1.$$

Recall the construction of G_2. There are

$$\min\left\{ l_i\hat{N}, \sum_{1\le k\le m_{b_i}} \sum_{t\in\mathcal{W}_k^{b_i},[a_i,d_i]\cap\mathcal{W}_k^{b_i}\neq\emptyset} \lfloor z^{b_i}_{t,k}\cdot\hat{N}\rfloor \right\}$$

edges in $\mathcal{E}_2$ covering task i.

There are at most λ_i slots in the interval $[a_i, d_i]$ to execute the tasks belonging to group b_i fractionally in the LP solution, and each slot belongs to at most two windows corresponding to the group b_i. Therefore, there are at most $2\lambda_i$ edges $u_t v^{b_i}_k$ in $\mathcal{E}_0$ with $z^{b_i}_{t,k} > 0$ and $t \in [a_i, d_i], 1 \le k \le m_{b_i}$, i.e., $\sum_{1\le k\le m_{b_i}} |\{t : t \in \mathcal{W}^{b_i}_k \cap [a_i, d_i]\}| \le 2\lambda_i$. The total weight of edges covering task i is

$$\frac{w_i}{l_i} \cdot \min\left\{ l_i\hat{N}, \sum_{k=1}^{m_{b_i}} \sum_{t\in\mathcal{W}_k^{b_i}\cap[a_i,d_i]} \lfloor z^{b_i}_{t,k}\cdot\hat{N}\rfloor \right\}$$

$$
\begin{aligned}
&\geq \frac{w_i}{l_i} \cdot \min\left\{ l_i\hat{N}, \sum_{k=1}^{m_{b_i}} \sum_{t\in\mathcal{W}_k^{b_i}\cap[a_i,d_i]} (z_{t,k}^{b_i}\cdot\hat{N}-1) \right\} \\
&\geq \frac{w_i}{l_i} \cdot \min\left\{ l_i\hat{N}, \left(\sum_{k=1}^{m_{b_i}} \sum_{t\in\mathcal{W}_k^{b_i}\cap[a_i,d_i]} z_{t,k}^{b_i}\cdot\hat{N} \right) - 2\lambda_i \right\} \\
&\geq \frac{w_i}{l_i} \cdot f^*\hat{N} - 2\lambda_i \cdot \frac{w_i}{l_i}.
\end{aligned}
$$

where the last inequality is because of (3.12) and $l_i \geq \gamma_i^*$.

It follows from Lemma 3.4 that there are at most $2\hat{N}-1$ colors in G_2. For each color $c, 1 \leq c \leq 2\hat{N}-1$, let $\bar{\mathcal{E}}_c$ denote the set of edges in $\mathcal{E}_2$ colored by c. We have

$$
\begin{aligned}
\sum_{c=1}^{2\hat{N}-1} \sum_{e\in\bar{\mathcal{E}}_c} w_e =& \sum_{i\in\mathcal{N}} \frac{w_i}{l_i} \cdot \min\left(l_i\hat{N}, \sum_{i\in\mathcal{N}} \sum_{k=1}^{m_{b_i}} \sum_{t\in\mathcal{W}_k^{b_i}\cap[a_i,d_i]} \lfloor z_{t,k}^{b_i}\cdot\hat{N} \rfloor \right) \\
\geq& \sum_{i\in\mathcal{N}} \left(\frac{f^*w_i}{l_i}\cdot\hat{N} - 2\lambda_i\cdot\frac{w_i}{l_i} \right) \\
=& \hat{N}\left(\sum_{i\in\mathcal{N}} \frac{f^*w_i}{l_i} - \sum_{i\in\mathcal{N}} \frac{2w_i\lambda_i/l_i}{\lambda^{1+\epsilon}} \right) \\
\geq& \hat{N}\left(\sum_{i\in\mathcal{N}} \frac{f^*w_i}{l_i} - \frac{2OPT}{\lambda^\epsilon} \right) \\
\geq& \hat{N}\left(1 - \frac{2}{\lambda^\epsilon} \right) OPT
\end{aligned}
$$

The second inequality follows from $\sum_{i\in\mathcal{N}} \frac{2w_i\lambda_i/l_i}{\lambda^{1+\epsilon}} \leq \frac{\max_{i\in\mathcal{N}} 2w_i/l_i}{\lambda^\epsilon} \leq \frac{2OPT}{\lambda^\epsilon}$ since executing only the most valued task is a trivial feasible scheduling. The last inequality follows from that the value of an optimal fractional LP solution is an upper bound on the value of the feasible scheduling.

By convexity, the set of edges of the same color with maximum weight before pruning, denoted by $\hat{\mathcal{E}}^*$, satisfy

$$
\sum_{e\in\hat{\mathcal{E}}^*} w_e \geq \frac{\hat{N}}{2\hat{N}-1}\left(1-\frac{2}{\lambda^\epsilon}\right) OPT > \frac{1}{2}\left(1-\frac{2}{\lambda^\epsilon}\right) OPT
$$

We then show the relationship between $\hat{\mathcal{E}}$ output by Algorithm 2 and $\hat{\mathcal{E}}^*$. For each task $i \in \mathcal{N}$, it follows from Step 5 that at most one edge in $\hat{\mathcal{E}}^*$ covering task i is removed from $\hat{\mathcal{E}}^*$, and the weight of the edge covering i is $\frac{w_i}{l_i}$, which is at most $\frac{1}{l_i+1}$ of the total

weight of the edges in $\hat{\mathcal{E}}^*$ covering task i. Therefore, it holds that

$$\begin{aligned}\sum_{e\in\hat{\mathcal{E}}} w_e &\geq W(\hat{\mathcal{E}}^*) - \frac{1}{l_i+1}W(\hat{\mathcal{E}}^*)\\ &\geq \frac{1}{2}(1-\frac{1}{l_i+1})\left(1-\frac{2}{\lambda^\epsilon}\right)OPT\\ &\geq \frac{1}{2}(1-\frac{1}{l_{min}+1})\left(1-\frac{2}{\lambda^\epsilon}\right)OPT\\ &= \frac{l_{min}}{2(l_{min}+1)}\left(1-\frac{2}{\lambda^\epsilon}\right)OPT,\end{aligned}$$

leading to an asymptotic approximation factor of $l_{min}/2(l_{min}+1)$.

Chapter 4

Downlink Transmission Scheduling with Data Sharing

This chapter formulates and analyzes a fundamental downlink transmission scheduling problem, which is a non-trivial extension of the baseline scenario of batching task scheduling problem, in a wireless communication system, composed of a base station, a set of transmission strategies and a set of users, each requesting a packet to be served within a time window. Some packets are requested by several users and can be served simultaneously due to the broadcast nature of the wireless medium. Each request can be served by a subset of transmission strategies, and requests need to be served in the FIFO model. We seek a downlink transmission scheduling algorithm maximizing the overall system utility. The above problem is termed as *downlink transmission scheduling with data sharing problem*.

Compared with existing works on the broadcast scheduling, batching task scheduling and the downlink transmission scheduling problems in Sections 2.1 to 2.3, the downlink transmission scheduling with data sharing problem has a stronger combinatorial flavor, as the scheduler needs to decide which subset of users to serve and under which transmission strategy. Moreover, the scheduling policy needs to comply with the FIFO model. Therefore, a dedicated algorithmic framework is called for which cannot be built on the existing models and approaches.

In this chapter, we develop an algorithmic framework of our downlink transmission scheduling with data sharing problem in both offline and online settings. We first establish its hardness by proving that (1) the offline problem is NP-hard, (2) the online problem is inapproximable in its generic form. Given the hardness result, we then develop approximation algorithms with mathematically proven performance guarantee.

We further conduct numerical analysis to evaluate the performance of our approximation algorithms.

4.1 Introduction

Consider the following fundamental downlink transmission scheduling problem in a generic wireless communication system composed of a base station and a set of users. Each user requests a packet, which needs to be served within a time window. Some packets are requested by several users, thus creating the opportunity for serving them simultaneously by a single transmission due to the broadcast nature of the wireless medium. The base station can choose from a set of transmission strategies, e.g., in terms of combination of data rate and coding scheme, to serve the users, by taking into account the data sharing opportunities. Each request can be served by a subset of transmission strategies, e.g., those satisfying the SNR constraint related to the user. When a user is served, a strategy-dependent utility is generated to the system. The problem faced by the base station is to design a downlink transmission scheduling algorithm maximizing the overall system utility.

The above downlink transmission scheduling problem significantly generalizes the canonical broadcast scheduling problem [12,18,24], in which the base station can choose only a single transmission strategy. Our problem is intuitively more challenging due to the following two factors. Firstly, due to data sharing, we need to decide which subset of users to serve, and under which transmission strategy; this is by nature a combinatorial optimization problem which is notoriously difficult to solve. Secondly, the potential data sharing opportunity further accentuates the combinatorial flavor, as we need to decide whether to seize the opportunity, at the price of delaying the transmission of the shared packet and also the subsequent packets. From a communication system perspective, our problem involves two intertwined sub-problems: (1) *admission control*, i.e., which user requests to serve? (2) *transmission optimization*, i.e., which transmission strategy to use for each request?

Driven by the above design challenges, we embark in this chapter on an algorithmic study of the above generic downlink transmission scheduling problem. We investigate both *offline* and *online* settings. In both settings, we establish the hardness of the scheduling problem by proving that (1) the offline problem is NP-hard, (2) the online problem is inapproximable in its generic form. Given the hardness result, we then focus on developing approximation algorithms with mathematically proven performance

guarantee in terms of approximation and competitive ratios, respectively.

- In the offline setting, we devise an algorithmic framework by constructing a graph, termed as *request graph*, and mapping the offline problem to the MWIS problem in the constructed graph by integrating the specific constraints posed by our problem; we then solve the LP relaxation of the IS problem in the graph; we further develop a coloring-based algorithm that rounds the solution of LP relaxation to an integer solution mapping to a feasible scheduling policy; we mathematically establish its performance bound.

- In the online setting, we devise an iterative algorithm that greedily selects the most profitable requests and serves them under a least robust strategy covering the request; we prove that, as long as each request has a certain slackness (cf. Section 4.4 for details), our algorithm can achieve a finite competitive ratio.

We further complement our theoretical analysis with numerical simulations that demonstrate the effectiveness of our algorithms in a variety of system settings.

The rest of this chapter is organized as follows. We formally state our downlink transmission scheduling problem in Section 4.2. In Section 4.3, we analyze the offline case and devise the scheduling algorithmic framework. In Section 4.4, we investigate the online case and present the design of an online scheduling algorithm. Section 4.5 conducts the simulation analysis evaluating the performance of our proposed algorithms. Section 4.6 concludes the chapter.

4.2 System Model and Problem Formulation

4.2.1 System Model

We consider a time-slotted wireless communication system composed of a base station and a set $\mathcal{N}$ of N users. We focus on the downlink transmission from the base station to users. Specifically, each user requests a packet from the base station[1]. Each request i ($1 \leq i \leq N$) is characterized by a couple (a_i, d_i), where a_i denotes the arrival time of the request at the base station, d_i denotes its deadline, i.e., request i needs to be finished by time d_i if it is served, $d_i - a_i + 1$ denotes the slackness of i. Let b_i denote the packet requested by user i. A packet may be requested by multiple users,

[1]Our analysis extends straightforwardly to the case where a user can request more than one packets by creating duplicates of the user and regarding each duplicated user as a new one.

creating potential data sharing opportunities due to the broadcast nature of the wireless medium. Throughout our analysis, we use the terms user and request interchangeably. There are B packets, indexed from 1 to B, in the base station. We normalize the slot duration to 1 for notation conciseness and regard all the time instances such as a_i and d_i as integers, otherwise we can perform a straightforward rounding process. Table 4.1 lists the main notations in this chapter.

Table 4.1: Main notations

$\mathcal{N}$	user set
N	number of users in $\mathcal{N}$, $N = \lvert\mathcal{N}\rvert$
T	time horizon in number of time slots
$\mathcal{R}$	set of transmission strategies in the base station
R	number of strategies in $\mathcal{R}$, i.e., $R = \lvert\mathcal{R}\rvert$
B	total number of packets
a_i	arrival time of user i
d_i	deadline of user i
b_i	packet requested by user i
$w_{i,r}$	reward of request i under strategy r
$\mathcal{R}_i$	set of transmission strategies covering request i
r_i^*	least robust strategy in $\mathcal{R}_i$
$\tau_{r,b}$	number of slots to transmit packet b under strategy r
$\mathcal{N}_b$	set of users requesting packet b
$\mathcal{N}_{b,t}$	set of users in $\mathcal{N}_b$ active at slot t
$\mathcal{N}_{b,t}^+(i,r)$	set of active users in $\mathcal{N}_b$ at slot t under r whose index $\geq i$
$\mathcal{N}_{b,t}^-(i,r)$	set of active users in $\mathcal{N}_b$ at slot t under r whose index $\leq i$
$\mathcal{N}_{b,t}(i,j;r_1,r_2)$	$\mathcal{N}_{b,t}^+(i,r_1) \setminus \mathcal{N}_{b,t}^+(j,r_2)$
G	request graph
$\mathcal{V}$	set of vertices in G
$\mathcal{E}$	set of edges in G
w_v	weight of vertex v
R_i	$\lvert\mathcal{R}_i\rvert$
M	$\max_{1\leq b\leq B} \lvert\mathcal{N}_b\rvert$
δ_i	$d_i - a_i - \min_{r\in\mathcal{R}_i} \tau_{r,b_i} + 2$
δ	$\sum_{i\in\mathcal{N}} \delta_i$
β_i	priority under which request i is served
β_v	priority of vertex v, a lower β value indicates higher priority
t_i	starting time to serve request i
t_v	starting time of the interval corresponding to v
r_i	strategy under which request i is served
r_v	strategy corresponding to v
l_v	length of the interval corresponding to v
y_v	binary variable indicating whether v is selected in the IS

To improve transmission reliability and efficiency, the base station typically adapts

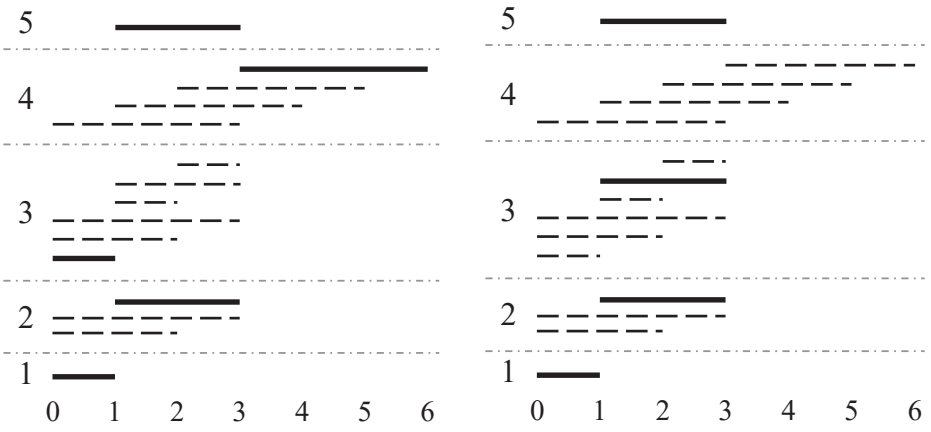
its transmission parameters, e.g., data rate, coding schemes, etc. To make our analysis generic without relying on any particular system, we consider the setting where the base station disposes a set $\mathcal{R}$ of R transmission strategies (e.g., in terms of combination of data rate and coding scheme), from which it may choose. For a given transmission strategy $r \in \mathcal{R}$, the transmitted packet can be decoded at user i if the SNR perceived by the user reaches a threshold. In this case, we say that user i can be reached or covered by strategy r. Let $\mathcal{R}_i$ denote the set of transmission strategies covering request i. For each strategy $r \in \mathcal{R}$ and each packet $b \in \mathcal{B}$, we denote $\tau_{r,b}$ the time (in number of slots) to transmit packet b under strategy r. A more robust strategy can reach more users at the price of longer transmission time under a lower but more robust data rate. Hence, any user reached by a strategy can also be reached by a more robust strategy. For any pair of strategies r and r', where r is less robust than r', we denote $r \prec r'$. We consider a typical case where $\prec$ is a strict total order over $\mathcal{R}$ such that we can rank all the strategies $r_1 \prec r_2 \prec \cdots \prec r_R$, and hence $\tau_{r_1,b} < \tau_{r_2,b} < \cdots \tau_{r_R,b}$ for each packet b. For each request i, let r_i^* denote the least robust strategy in $\mathcal{R}_i$. If request i is served under strategy r, a reward $w_{i,r}$ is generated to the system. Under the above generic model, we are interested in seeking an optimal scheduling policy for the base station to maximize the total reward within a given time horizon T.

4.2.2 Problem Formulation

We are interested in both offline and online settings. In both cases the scheduler should follow the first-in-first-out (FIFO) service model such that admitted requests are queued at the scheduler in the same order they arrive and a request is cleared by the scheduler when the corresponding user is served.[2] Without loss of generality, we assume that $a_i \leq a_j$ for any $1 \leq i < j \leq N$. In case where $a_i = a_j$ and $i < j$, user i should be served first. Due to the broadcast nature of the wireless medium, the standard FIFO model needs to be tailored in our context.

Definition 4.1 (Adapted FIFO Model)**.** *Any scheduling policy satisfying the following requirement is called an adapted FIFO model. When starting serving request i under strategy r at slot t, any low-priority request $j > i$ requesting packet b_i must be simultaneously served if $r \in \mathcal{R}_j$ and $a_j \leq t \leq d_j - \tau_{r,b_i} + 1$.*

[2]In case of tie, a predefined rule is applied to determine the service order.



(a) Adapted FIFO model (b) Standard FIFO model

Figure 4.1: Illustration of our adapted FIFO model vs. standard FIFO model

Example 4.1. *To illustrate our adapted FIFO model against the standard FIFO model policy, we consider an example composed of five requests* $(0,1)$, $(0,3)$, $(0,3)$, $(0,6)$ *and* $(1,3)$, *indexed from 1 to 5, concerning the same packet, denoted by 1, where* $\mathcal{R} = \{1,2,3\}$, $\mathcal{R}_1 = \{1,2,3\}$, $\mathcal{R}_2 = \{2,3\}$, $\mathcal{R}_3 = \{1,2,3\}$, $\mathcal{R}_4 = \{3\}$, $\mathcal{R}_5 = \{2,3\}$, *and* $\tau_{1,1} = 1, \tau_{2,1} = 2$ *and* $\tau_{3,1} = 3$. *For each request* $i \in \{1,2,3,4,5\}$ *and each strategy* $r \in \mathcal{R}_i$, *the reward of request* i *under strategy* r *is 1, i.e.,* $w_{i,r} = 1$. *The lines in Figure 4.1 illustrate the possible executions of the requests, with the length being the requested execution time.*

- *Figure 4.1(a) illustrates the optimal scheduling policy under the adapted FIFO model, which starts serving requests 1 and 3 under strategy 1 at slot 0, requests 2 and 5 under strategy 2 at slot 1, request 4 under strategy 3 at slot 3. All the five requests are served.*

- *Figure 4.1(b) illustrates the optimal scheduling policy under the standard FIFO model, which starts serving request 1 under strategy 1 at slot 0, requests 2, 3 and 5 under strategy 2 at slot 1, while rejecting request 4, or starts serving request 1 under strategy 1 at slot 0, requests 2 and 3 under strategy 2 at slot 1, request 4 under strategy 3 at slot 3, while rejecting request 5. Different from our adapted model, a request, if admitted, should be started no later than any admitted request arriving later. We can check that under the standard FIFO model, at most four requests can be served.*

We remark that (1) the adapted FIFO model takes into account the packet transmission and is thus more flexible and naturally leads to better efficiency, as demonstrated

by the illustration example, (2) it is technically more involved than the standard FIFO model. Our mathematical framework can be adapted if the standard FIFO model is employed.

In the adapted FIFO model, for a number of requests served simultaneously, among which the highest priority request is i, we say that these requests are served, or *batched*, under priority i, and we call i the *head* of the batched requests. In the degenerated case where i is served alone, its batch contains only itself, which is also the batch header.

We proceed by defining the *feasible scheduling policy* for downlink transmission scheduling, concisely termed as feasible policy in this chapter. We say that a feasible policy is optimal if it maximizes the system utility.

Definition 4.2 (Feasible Scheduling Policy for Downlink Transmission Scheduling)**.** *We call a policy feasible if the following conditions are met.*

- *each request is served at most once;*
- *the base station does not transmit more than one packet simultaneously;*
- *the policy is an adapted FIFO model.*

4.3 The Offline Case

In this section, we consider the offline downlink transmission scheduling problem, denoted by **P3**. We refer the readers to Appendix 4.7 for a detailed integer linear programming formulation of **P3**. In the following theorem, we prove that **P3** is NP-hard. To make our presentation more streamlined, we provide the proof sketch of the lemmas and theorems in the main text and assemble the detailed proof in Appendix 4.7.

4.3.1 Problem Hardness

Theorem 4.1 (Hardness of **P3**)**.** *The offline scheduling problem **P3** is NP-hard. Its discrete version is NP-complete.*

Proof Sketch. We prove that the classical 0-1 Knapsack problem can be reduced to **P3**. It then follows from the NP-hardness of the 0-1 Knapsack problem that **P3** is NP-hard. It follows from the NP-completeness of Knapsack problem with integer weights that the discrete version of **P3** is also NP-complete. □

Given the hardness of **P3**, we devote our efforts to designing our approximation scheduling algorithm. Our central idea is to construct a graph, termed as *request graph*, to capture the relationships among requests, given the users' requests and the constraint imposed by the adapted FIFO model. We then cast **P3** to the MWIS problem in the constructed request graph.

This section is organized as follows. We first construct the request graph and cast **P3** to the MWIS problem in the graph. We then explore the structural properties of the constructed request graph to formulate the LP relaxation of **P3**. We further develop an approximation algorithm based on rounding the solution of the LP relaxation to an integer solution. The approximation ratio of our approximation algorithm is established to complete the section.

4.3.2 Request Graph

We start with a few definitions to simplify subsequent presentation. For any slot t and each strategy r, we call a user i *active* at t under r if $r \in \mathcal{R}_i$ and $a_i \leq t \leq d_i - \tau_{r,b_i} + 1$. Intuitively, when starting serving user i at slot t under strategy r, the corresponding request can be finished by its deadline d_i. We call a user active at slot t if there exists at least one strategy $r \in \mathcal{R}$, under which the user is active, and we call slot t an *active* slot. Similarly, we call the packet b *active* at slot t if there is a user requesting packet b that is active at slot t. For any packet b, let $\mathcal{N}_b$ denote the set of users requesting packet b and $\mathcal{N}_{b,t}$ denote the set of users in $\mathcal{N}_b$ active at slot t. For any user $i \in \mathcal{N}_{b,t}$ and any strategy r, let $\mathcal{N}_{b,t}^{+}(i,r)$ denote the set of active users in $\mathcal{N}_b$ at t under r whose indexes $\geq i$, i.e., whose priorities are lower than or equal to that of i. Mathematically

$$\mathcal{N}_{b,t}^{+}(i,r) \triangleq \{k : k \in \mathcal{N}_{b,t}, k \geq i, a_k \leq t \leq d_k - \tau_{r,b} + 1\}.$$

Similarly, define

$$\mathcal{N}_{b,t}^{-}(i,r) \triangleq \{k : k \in \mathcal{N}_{b,t}, k \leq i, a_k \leq t \leq d_k - \tau_{r,b} + 1\}.$$

For any pair of users $i, j \in \mathcal{N}$ and any pair of strategies $r_1, r_2 \in \mathcal{R}$, we further define

$$\mathcal{N}_{b,t}(i,j;r_1,r_2) \triangleq \mathcal{N}_{b,t}^{+}(i,r_1) \setminus \mathcal{N}_{b,t}^{-}(j,r_2).$$

The request graph, denoted by $G \triangleq (\mathcal{V}, \mathcal{E})$, consists of the following vertices and

edges.

4.3.2.1 Vertices.

Each vertex in $\mathcal{V}$ maps to a time interval, a transmission strategy, and a priority, and each vertex covers a set of users. Consider each active slot t, each packet b corresponding to an active user at t, each user $i \in \mathcal{N}_{b,t}$, and each strategy $r \in \mathcal{R}_i$. We create the following two types of vertices.

- **Type-1 vertices.** For each user $j \in \mathcal{N}_{b,t}$ with $i < j$, and each strategy $r' \in \mathcal{R} \backslash \mathcal{R}_i$, we create a vertex, denoted by v, mapping to the time interval $[t, t + \tau_{r,b} - 1]$, the strategy r, and the priority i. Physically, the constructed vertex v corresponds to a possible execution of the requests in $\mathcal{N}_{b,t}(i, j; r, r')$ under strategy r. We say that v covers the requests in $\mathcal{N}_{b,t}(i, j; r, r')$. If the scheduler decides to start transmitting packet b under strategy r at slot t to serve the requests covered by v, we say that the requests in $\mathcal{N}_{b,t}(i, j; r, r')$ are *instantiated* by v. We define a weight w_v for v and set $w_v \triangleq \sum_{k \in \mathcal{N}_{b,t}(i,j;r,r')} w_{k,r}$.

- **Type-2 vertices.** We create a vertex v mapping to the time interval $[t, t+\tau_{r,b}-1]$, the strategy r, and the priority i. It follows from the same notations as type-1 vertices that v covers the requests in $\mathcal{N}^+_{b,t}(i, r)$. We define its weight as $w_v \triangleq \sum_{k \in \mathcal{N}^+_{b,t}(i,r)} w_{k,r}$.

We use $\mathcal{V}_i$ to denote the set of vertices covering user i. For each constructed vertex v, let t_v and l_v denote the starting time and the length of time interval corresponding to v, let r_v denote the strategy of v, and let β_v denote the priority of v. Note that a smaller β value indicates a higher priority.

Example 4.2. *Figure 4.2 illustrates the vertex construction for the example in Figure 4.1(a). The left subfigure replots the possible cases to serve each request independently. The right subfigure illustrates the created vertices, more precisely specified in Table 4.2. The unique optimal scheduling policy, starting serving users* 1 *and* 3 *under strategy* 1 *at slot* 0*, users* 2 *and* 5 *under strategy* 2 *at slot* 1*, user* 4 *under strategy* 3 *at slot* 3*, maps to vertices* v_1*,* v_{10}*, and* v_{17}*.*

We now analyze the complexity of creating the vertices in G. Denote $\delta_i \triangleq d_i - a_i - \min_{r \in \mathcal{R}_i} \tau_{r,b_i} + 2$ for each user i requesting the packet b, and $\delta \triangleq \sum_{i \in \mathcal{N}} \delta_i$. Let $M \triangleq \max_{1 \le b \le B} |\mathcal{N}_b|$. There are at most $\delta_i(R - |\mathcal{R}_i|)|\mathcal{N}_b|$ type-1 vertices and $\delta_i|\mathcal{R}_i|$

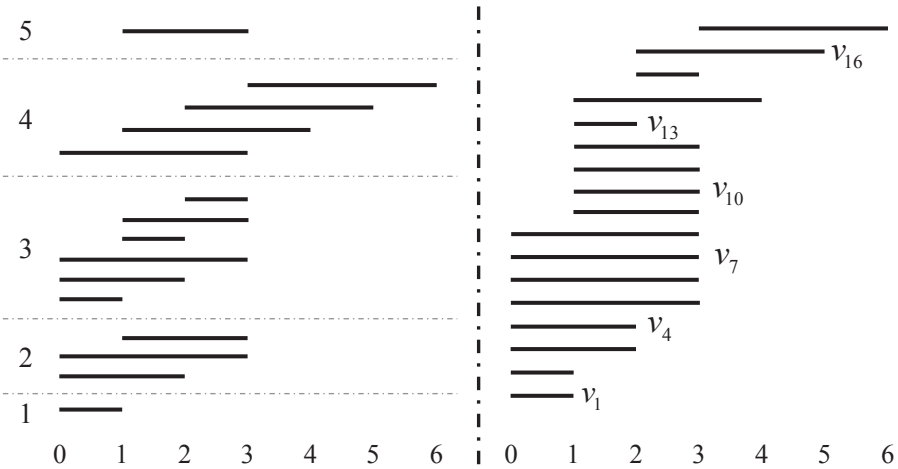


Figure 4.2: Illustration of vertex construction for Example 4.2

Table 4.2: Constructed vertices in Example 4.2

Vertex	Covered user(s)	Priority	Time interval	Strategy
v_1	1,3	1	[0,1)	1
v_2	3	3	[0,1)	1
v_3	2,3	2	[0,2)	2
v_4	3	3	[0,2)	2
v_5	2,3,4	2	[0,3)	3
v_6	2,4	2	[0,3)	3
v_7	3,4	3	[0,3)	3
v_8	4	4	[0,3)	3
v_9	2,3,5	2	[1,3)	2
v_{10}	2,5	2	[1,3)	2
v_{11}	3,5	3	[1,3)	2
v_{12}	5	5	[1,3)	2
v_{13}	3	3	[1,2)	1
v_{14}	4	4	[1,4)	3
v_{15}	3	3	[2,3)	1
v_{16}	4	4	[2,5)	3
v_{17}	4	4	[3,6)	3

type-2 vertices whose priorities are assigned the priority level i. It then holds that the total number of vertices sums up to $O(\delta RM)$. Listing users in $\mathcal{N}_b, \forall 1 \leq b \leq B$ takes $O(N)$ time. Consider each packet b and each corresponding active slot t. Among the users in $\mathcal{N}_b$, listing the set of users in $\mathcal{N}_{b,t}$ takes $O(|\mathcal{N}_b|)$ time. For each strategy r, among the users in $\mathcal{N}_{b,t}$, listing the set of active users at t under r takes $O(|\mathcal{N}_{b,t}|)$ time. For all pair of users $i, j \in \mathcal{N}_{b,t}$ and all pair of r, r' with $r \in \mathcal{R}_i$ and $r' \in \mathcal{R} \setminus \mathcal{R}_i$, further listing users in $\mathcal{N}_{b,t}(i, j; r, r')$ and $\mathcal{N}^+_{b,t}(i, r)$ takes $O(|\mathcal{N}_b|^2(R - |\mathcal{R}_i|)|\mathcal{R}_i|) = O(M^2R^2)$ time. For each packet b, there are at most $\sum_{k\in\mathcal{N}_b} \delta_k$ slots at which packet b is active since there are δ_k slots at which each request k is active. Therefore, the construction of the vertices can be completed in $O(\sum_{1\leq b\leq B} \sum_{k\in\mathcal{N}_b} \delta_k M^2R^2) = O(\delta M^2R^2)$ time.

4.3.2.2 Edges.

The edges in G can be categorized into the following three types.

Inter-user edges (type-1 and type-2). For any pair of vertices u and v not covering any common user, we distinguish the following two subcategories.

- **Type-1 edges.** We construct an edge uv if the time intervals corresponding to u and v overlap each other, indicating the impossibility of instantiating the requests, which are covered by u, by u and also the requests, which are covered by v, by v.

- **Type-2 edges.** This type of edges are further classified into two cases, capturing our adapted FIFO model. Specifically, we construct an edge between any pair of nodes u and v if they satisfy the following conditions, with each set of conditions corresponding to a case.

  - $\beta_u < \beta_v$ and $t_u > t_v$, or $\beta_u > \beta_v$ and $t_u < t_v$.
  - (1) the users covered by u and v request a same packet, (2) $\beta_v < \beta_u, t_v \leq t_u$, and (3) there exists a request j covered by u with $r_v \in \mathcal{R}_j$ and $a_j \leq t_v \leq d_j - \tau_{r_v,b_j} + 1$, i.e., request j fits into the time interval corresponding to v. We clarify the implication of this case of edges via an example. Suppose the requests covered by v are instantiated by v. It follows from the adapted FIFO model that j must be served under the priority β_v or batched with another request with higher priority than β_v. Hence, the requests covered by v and request j cannot be both instantiated by v and u, respectively.

Intra-user edges (type-3). We construct an edge between each pair of vertices

covering at least a same user, i.e., $\forall u, v \in \mathcal{V}_i, uv \in \mathcal{E}, \forall i \in \mathcal{N}$. The intra-user edges model the constraint that any user is served at most once.

We then analyze the complexity of creating the edges in G. Noticing that there are at most M requests in $\mathcal{N}_b, 1 \leq b \leq B$, it follows from the definitions of $\mathcal{N}_{b,t}(i, j; r, r')$ and $\mathcal{N}^+_{b,t}(i, r)$ that each constructed vertex covers at most M requests. Therefore, for each pair of vertices, checking whether the condition for the second case of type-2 edges is met takes $O(M)$ time. As the number of vertices in G is $O(\delta RM)$, the construction of the edges in G can be completed in $O(\delta^2 R^2 M^3)$ time.

4.3.3 From Downlink Transmission Scheduling to Maximum Weighted Independent Set

Armed with the constructed request graph G, we now prove that our offline scheduling problem can be cast to the MWIS problem of G. The following lemma establishes the relationship between the feasible policies and the ISes of G. By choosing an IS in G we mean to serve the requests covered by the vertices in the chosen IS.

Lemma 4.1. *Each IS of G maps to a feasible policy. Each optimum feasible policy maps to an IS of G.*

Proof Sketch. The proof of each IS mapping to a feasible policy follows from the construction of G, where each type of edges characterizes a constraint of the feasible scheduling policy. To prove that each optimal feasible policy maps to an IS of G, we consider an optimal feasible policy, and seek a subset of vertices in $\mathcal{V}$ covering all requests served by the optimal feasible policy. Based on the vertex construction, we prove that the subset of vertices is an IS. Therefore, each optimal feasible policy maps to an IS. □

Lemma 4.1 immediately leads to the following corollary.

Corollary 4.1. ***P3*** *can be cast to the MWIS problem, formulated below.*

$$\begin{aligned} \textit{maximize} \quad & \sum_{v \in \mathcal{V}} w_v y_v \\ \textit{subject to} \quad & y_v + y_u \leq 1, \quad \forall uv \in \mathcal{E} \\ & y_v \in \{0, 1\}, \quad \forall v \in \mathcal{V}, \end{aligned} \tag{4.1}$$

where y_v is the binary variable indicating whether v is selected in the IS. Constraint (4.1) *states that any pair of neighboring vertices cannot be chosen together.*

4.3.4 LP Relaxation

To solve the MWIS problem formulated above, we first solve the linear programming (LP) relaxation of MWIS problem and then round the fractional solution of LP relaxation to an IS. In this subsection, we develop an LP relaxation approach tailored to our problem. The next subsections are devoted to the rounding technique and the mathematical performance analysis. Table 4.3 lists the additional notations used in Sections 4.3.4 to 4.3.6.

Table 4.3: Additional notations in Sections 4.3.4 to 4.3.6

$\mathcal{V}^{\alpha}(i)$	set of vertices $v \in \mathcal{V}$ with $\beta_v \leq i$ and $t_v \geq a_i$
$\mathcal{N}^{\alpha}(i)$	set of users containing at least one vertex in $\mathcal{V}^{\alpha}(i)$
φ	$\max_{k \in \mathcal{N}}(d_k - a_k + 1)$
l_{min}	$\min_{1 \leq b \leq B} \min_{r \in \mathcal{R}} \tau_{r,b}$
l_{max}	$\max_{1 \leq b \leq B} \max_{r \in \mathcal{R}} \tau_{r,b}$
$\hat{N}$	$RM\delta^{1+\epsilon}$
y_v^*	value of y_v in the solution of LP relaxation
$\mathcal{C}$	ordered set (or vector) of colors

In the LP relaxation, for each vertex $v \in \mathcal{V}$, we replace the constraint $y_v \in \{0, 1\}$ by $y_v \geq 0$. There is no need to explicitly add the constraint $y_v \leq 1$ since it is implied by the other constraints. We add the constraints $\sum_{v \in \mathcal{V}_i} y_v \leq 1$ indicating that each request i is served at most once. It is well-known that the LP relaxation of the MWIS problem suffers the so-called *half integer* effect due to the edge constraint [39]. To mitigate this effect, we construct the following two types of constraints to replace the edge constraint (4.1), where each type concerns a particular type of inter-user edges constructed in G, and thus is equivalent to the specific constraint of our scheduling problem.

Type-1 constraints. The first type of constraints, as shown in (4.2), concerns the vertices in $\mathcal{V}$ and type-1 edges. For each time slot $t \in [0, T]$, we call a vertex v *active* at slot t if $t \in [t_v, t_v + l_v - 1]$. The intuition behind the definition of active vertex is that if the corresponding requests are instantiated by an active vertex at slot t, then the requests are still in serving at slot t. Recall the construction of the request graph G, the vertices in $\mathcal{V}$ that are active at slot t are adjacent to each other. Type-1 constraints

state that at most one active vertex at slot t is chosen at any feasible scheduling policy.

$$\text{Type-1 constraints:} \sum_{v\in\mathcal{V}, v \text{ is active at slot } t} y_v \leq 1, \quad \forall t \in [0, T]. \tag{4.2}$$

Let $l_{min} \triangleq \min_{1\leq b\leq B} \min_{r\in\mathcal{R}} \tau_{r,b}$ and $l_{max} \triangleq \max_{1\leq b\leq B} \max_{r\in\mathcal{R}} \tau_{r,b}$. For each slot $t' \in [0, l_{min} - 2]$, the vertices active at slot t' are also active at $l_{min} - 1$. Similarly, for each slot $t' \in [T-l_{min}+2, T]$, each vertex active at slot t' is also active at slot $T-l_{min}+1$. Therefore, we just need to list type-1 constraints for slots $t \in [l_{min} - 1, T - l_{min} + 1]$, and each slot corresponds to at most one type-1 constraint. Hence, there are at most $T - 2l_{min} + 1$, asymptotically $O(T)$, type-1 constraints.

Type-2 constraints. The second type of constraints, as shown in (4.3), concerns the vertices in $\mathcal{V}$ and type-2 edges. For each time slot t, we call a vertex v *valid* at slot t if $t_v \geq t$. Any request instantiated by a valid vertex v at t does not start being served earlier than t. For any user i, let $\mathcal{V}^\alpha(i)$ denote the set of vertices $v \in \mathcal{V}$ satisfying: (1) the priority of v is higher than or equal to the priority of i, i.e., $\beta_v \leq i$, (2) v is valid at slot a_i, i.e., $t_v \geq a_i$. Mathematically, $\mathcal{V}^\alpha(i) \triangleq \{v : v \in \mathcal{V}, \beta_v \leq i, t_v \geq a_i\}$. Physically, the set of vertices in $\mathcal{V}^\alpha(i)$ includes all vertices $v \in \mathcal{V}$ satisfying: (1) the priority of v is higher than or equal to the priority of i, (2) v is connected with a vertex in $\mathcal{V}_i$ by a type-2 edge. Let $\mathcal{N}^\alpha(i)$ denote the set of users containing at least one vertex in $\mathcal{V}^\alpha(i)$.

For each request i, any feasible scheduling policy can transmit at most $\max_{k\in\mathcal{N}^\alpha(i)}(d_k - a_i + 1)/l_{min}$ packets during the time interval $[a_i, \max_{k\in\mathcal{N}^\alpha(i)} d_k]$ because the base station does not transmit multiple packets simultaneously. Therefore, there are at most $\max_{k\in\mathcal{N}^\alpha(i)}(d_k - a_i + 1)/l_{min}$ vertices in $\mathcal{V}^\alpha(i)$ at any feasible scheduling policy, leading to the following constraint.

$$\text{Type-2 constraints:} \sum_{v\in\mathcal{V}^\alpha(i)} y_v \leq \frac{\max_{k\in\mathcal{N}^\alpha(i)}(d_k - a_i + 1)}{l_{min}}, \quad \forall i \in \mathcal{N}. \tag{4.3}$$

We now derive the complexity for establishing type-2 constraints. For each user i, there is at most one type-2 constraint. The complexity for listing all vertices in $\mathcal{V}^\alpha(i)$ is $O(\sum_{k\in\mathcal{N}_b, k\leq i} \delta_k RM)$. Hence, listing type-2 constraints takes $O(\sum_{i\in\mathcal{N}} \sum_{k\in\mathcal{N}_b, k\leq i} \delta_k RM) = O(M^2 R \sum_{k\in\mathcal{N}} \delta_k) = O(M^2 R\delta)$ time.

We now replace the edge constraint (4.1) by the above constructed constraints to

formulate the LP relaxation of the MWIS problem, denoted by **P3'**, as below.

$$\begin{aligned}
\textbf{P3':} \quad \text{maximize} \quad & \sum_{v\in\mathcal{V}} w_v y_v \\
\text{subject to} \quad & (4.2), (4.3) \\
& \sum_{v\in\mathcal{V}_i} y_v \leq 1, \quad \forall i \in \mathcal{N} \\
& y_v \geq 0, \quad \forall v \in \mathcal{V}
\end{aligned}$$

By the analysis in this subsection, each optimal feasible scheduling policy is also a feasible solution of **P3'**. Hence, the value of an optimum fractional solution of **P3'** is an upper bound for the utility of the optimal feasible policy.

4.3.5 Approximation Scheduling Algorithm Design

Our approximation algorithm first solves **P3'** and then applies the rounding and coloring technique developed in [7] adapted to our context. To make our presentation streamlined and self-contained, we present and analyze the adapted rounding algorithm in the context of our problem. At a high level, our algorithm colors the graph such that each color induces an IS that can be mapped to a feasible scheduling policy. The algorithm essentially follows the procedures in [7], which are adapted below in the context of our problem. Some additional notations are used in this subsection. We define $\varphi \triangleq \max_{k\in\mathcal{N}}(d_k - a_k + 1)$. Let $\hat{N} = RM\delta^{1+\epsilon}$. To make our analysis concise, we assume that $\hat{N}$ is an integer, otherwise we need to round it to the nearest integer. Let $\mathcal{C}$ denote an ordered set (or a vector) of $(1 + \varphi/l_{min})\hat{N} - 1$ colors.

We first solve **P3'** and denote the solution by $\{y_v^*\}_{v\in\mathcal{V}}$, where y_v^* is the value of y_v in the LP relaxation. Clearly, we have $0 \leq y_v^* \leq 1, \forall v \in \mathcal{V}$. Then, we color the graph G such that (1) each vertex $v \in \mathcal{V}$ receives $\lfloor \hat{N} y_v^* \rfloor$ colors, and (2) any color used to color any vertex v is not used to color any neighbor of v. Technically, we sort the vertices in $\mathcal{V}$ non-decreasingly by their starting time with ties broken randomly. For each vertex $v \in \mathcal{V}$ from left to right, we use the first $\lfloor \hat{N} y_v^* \rfloor$ colors in $\mathcal{C}$ not used to color any neighbor of v to color v. The algorithm outputs the set of vertices sharing one common color with maximum weight.

The pseudo-code of our algorithm is given in Algorithm 3. As each vertex corresponds to a request, the final scheduling is to serve the requests corresponding to the set of vertices output by Algorithm 3. In the algorithm, the function **Color**$(v, y_v^*, \mathcal{C})$

returns the first $\lfloor \hat{N} y_v^* \rfloor$ colors in $\mathcal{C}$ that have not been used to color any neighbor of v. **Color** is a graph algorithm that can be coded straightforwardly. The detailed implementation is thus omitted in the pseudo-code.

Algorithm 3 Offline downlink transmission scheduling: executed by the scheduler

1: **Input:** request graph $G = (\mathcal{V}, \mathcal{E})$, solution of the LP relaxation $\{y_v^*\}_{v \in \mathcal{V}}$, $\hat{N} \leftarrow RM\delta^{1+\epsilon}$, a vector $\mathcal{C}$ of $(1 + \varphi/l_{min})\hat{N} - 1$ colors
2: **Output:** set of vertices ▷ The final scheduling policy is to serve requests corresponding to $\hat{\mathcal{V}}$
3: sort the vertices in $\mathcal{V}$ non-decreasingly by their starting time, breaking ties randomly
4: **for** each $v \in \mathcal{V}$ from left to right **do**
5: color vertex v using the colors **Color**$(v, y_v^*, \mathcal{C})$
6: **end for**
7: **return** set of vertices in $\mathcal{V}$ sharing one common color with maximum weight

4.3.6 Performance Analysis

In this subsection we derive the theoretical performance guarantee of our approximation algorithm. We first prove that each vertex $v \in \mathcal{V}$ with $\lfloor y_v^* \hat{N} \rfloor \geq 1$ is colored by $\lfloor y_v^* \hat{N} \rfloor$ colors in Lemma 4.2, based on which we are then able to establish the approximation factor of Algorithm 3 in Theorem 4.2. We conclude this subsection by giving the complexity of Algorithm 3.

Lemma 4.2. *Each vertex $v \in \mathcal{V}$ with $\lfloor y_v^* \hat{N} \rfloor \geq 1$ is colored by $\lfloor y_v^* \hat{N} \rfloor$ colors by Algorithm 3.*

Proof Sketch. We first derive the upper bound of the number of colors that have been used to the neighbors of v when Algorithm 3 colors v. Based on the number of colors disposed in $\mathcal{C}$, we can then derive that there are at least $\lfloor y_v^* \hat{N} \rfloor$ available colors that can be used to color v. □

Theorem 4.2. *Algorithm 3 outputs an asymptotically $1/(1 + \varphi/l_{min})-$optimal feasible scheduling policy for our offline scheduling problem, i.e., Algorithm 3 is a $1/(1 + \varphi/l_{min})-$approximation algorithm.*

Proof Sketch. It follows from Lemma 4.1 and Lemma 4.2 that we get at most $(1 + \varphi/l_{min})\hat{N}$ ISes of G. We prove that there exists an IS whose total weight is at least $1/(1 + \varphi/l_{min})$ of the utility for the optimal feasible scheduling policy. □

Theorem 4.2 demonstrates that the approximation factor of our algorithm only depends on the ratio between the largest task slackness and l_{min}, i.e., its performance does not degrade with the system size.

To conclude the analysis of our algorithm, we analyze its complexity. To that end, we firstly need to compute the number of vertices in $\mathcal{V}$. To that end, we can calculate that the number of created vertices in $\mathcal{V}$ is at most δRM. The number of colors received by each vertex is $O(\hat{N})$, and thus the complexity of the color process is $O(RM\delta\hat{N})$. The complexity of the sorting process is $O(RM\delta \log(RM\delta))$. Therefore, It follows from $\hat{N} = RM\delta^{1+\epsilon}$ that the complexity of our algorithm is $O(R^2M^2\delta^{2+\epsilon})$.

4.4 The Online Case

In this section, we consider the online scheduling problem, where the scheduler only knows the current scheduling backlog. More specifically, the parameters $a_i, d_i, b_i, \mathcal{R}_i$ for request i are known only at the moment of arrival. As the offline case, the online downlink scheduling problem also takes into account our adapted FIFO model. Table 4.4 lists the additional notations used in Section 4.4.

Table 4.4: Additional notations in Section 4.4

$\mathcal{T}$	set of slots at which Algorithm 4 starts serving at least one request
$\mathcal{O}$	set of requests that are served by the optimal scheduling policy
Ω_t	set of ready requests at slot t
$\mathcal{B}(\Omega_t)$	set of packets, each requested by at least one user in Ω_t
B_{max}	$\max_{1\leq t\leq T} \lvert\mathcal{B}(\Omega_t)\rvert$
$\mathcal{N}_t^*$	set of requests that start being served by Algorithm 4 at slot t

In the online setting, we focus on the *non-preemptive* scheduling model: once a request starts being served, it must be completed without interruption [17] [21]. We note that the non-preemptive model is seamlessly compatible with our adapted FIFO model.

4.4.1 Problem Inapproximability

We start by showing in the following theorem that the online problem in its generic form cannot be approximated with any finite competitive ratio.

Theorem 4.3. *For any $\rho > 0$, there exists an instance of our scheduling problem, where the competitive ratio of any deterministic online algorithm Π is larger than ρ.*

Proof. We consider the following instance of our online scheduling problem. There is only one packet and one strategy in the base station; the station needs 2 slots to transmit the packet under the strategy. In slot 0, a request with the couple $(0, 1)$ arrives, and its reward is 1 if it is served under the strategy. We distinguish two cases.

- If Π does not serve the request, then no more request arrives, leading to a competitive ratio of infinity as the optimal solution is clearly to serve the request;
- If Π serves the request, a set of requests arrive, each characterized by the same couple $(1, 2)$, and the total reward of them is larger than ρ if they are served under the strategy. Under this request sequence, the optimal scheduling policy serves the requests arriving at slot 1, while Π only serves the request arriving at slot 0, leading to a competitive ratio of larger than ρ.

Combining the above two cases completes our proof. □

4.4.2 Online Scheduling Algorithm Design

Theorem 4.3 establishes the inapproximability of the online downlink scheduling problem in the generic case. By examining the problem instance in Theorem 4.3, we observe that the inapproximability is significantly due to the stringent deadline. Consequently, an online algorithm is forced to make a decision immediately when the requests arrive. Motivated by this observation, we slightly relax the slackness of the requests, and design an online scheduling algorithm with bounded competitive ratio. Specifically, we assume that the slackness of each request i is at least $l_{max}+\min_{r\in\mathcal{R}_i}\tau_{r,b_i}-1$, i.e., $d_i - a_i + 1 \geq l_{max} + \min_{r\in\mathcal{R}_i}\tau_{r,b_i} - 1, \forall i \in \mathcal{N}$. Moreover, for each user i and each pair of strategies $r, r' \in \mathcal{R}_i$ with $r \prec r'$, we assume that $w_{i,r} \geq w_{i,r'}$.

We give a high-level overview of our online scheduling algorithm. For each slot t, at which there is no request being served, our algorithm starts serving a set of users at slot t requesting a same packet under the least robust strategy covering the users such that the total reward of the users is maximum among the packets.

Technically, we introduce the following definitions. We say a slot *free* if there is no request being served at the slot. For each slot t, we say a request k *ready* at slot t satisfying: (1) k is active at slot t, i.e., $a_k \leq t \leq d_i - \min_{r\in\mathcal{R}_i}\tau_{r,b_i} + 1$, (2) k has not been served before slot t, and (3) starting serving k at slot t does not violate the adapted FIFO model. Let Ω_t denote the set of ready requests at slot t, and let $\mathcal{B}(\Omega_t)$ denote the packets, each of which is requested by at least one user in Ω_t. We define

$B_{max} \triangleq \max_{1 \le t \le T} |\mathcal{B}(\Omega_t)|$. We use a quadruple (i, t_i, r_i, β_i) to denote each served request i, where t_i is the starting time to serve request i, r_i and β_i are the strategy and the priority under which i is served.

The pseudo-code of our algorithm is given in Algorithm 4. Consider the current slot t. If the slot t is not free or there is no ready request at slot t, i.e., $\Omega_t = \emptyset$, there is no request that starts being served at slot t; otherwise, for each packet $b \in \mathcal{B}(\Omega_t)$, we first seek the least robust strategy $\overline{r}_b$ that can reach all users in Ω_t requesting packet b, i.e., $\overline{r}_b = \text{argmin}_{r \in \mathcal{R}_i, i \in \Omega_t \cap \mathcal{N}_b} \tau_{r,b}$. We then seek the packet b such that the total reward of the users in $\Omega_t \cap \mathcal{N}_b$ under strategy $\overline{r}_b$ is maximum among the packets in $\mathcal{B}(\Omega_t)$. Let Γ denote the set of users in Ω_t requesting packet b. Finally, we start serving the users in Γ at slot t by transmitting packet b under the strategy $\overline{r}$ and the priority of request $\min_{k \in \Gamma} k$.

We briefly describe the following functions used in our algorithm, which can be coded straightforwardly. The detailed implementation is thus omitted in the pseudo-code.

- **Packets**(Ω) returns the set of packets, each of which is requested by at least one user in Ω.
- **UsersRequestPacket**(Ω, b) returns the set of users requesting packet b.
- **LeastRobustStrategy**(Ω) returns the least robust strategy $\overline{r}$ that can reach all requests in Ω, i.e., $\overline{r} = \text{argmin}_{r \in \mathcal{R}_i, i \in \Omega} \tau_{r,b}$.
- **TotalReward**(Ω, r) returns the total reward of requests in Ω under strategy r.
- **HighestPriorityRequest**(Γ) returns the highest priority request in Γ.

Algorithm 4 Online downlink transmission scheduling: executed by the scheduler at each slot t

Input: Ω_t: the set of ready requests at slot t
Output: $\mathcal{N}_t^*$, the set of requests started being served at slot t, with each admitted request i served under strategy r_i and priority β_i
Initialization: $\mathcal{N}_t^* \leftarrow \emptyset$
if $\Omega_t == \emptyset$ **then**
    return $\emptyset$
end if
$\mathcal{B}(\Omega_t) \leftarrow$ **Packets**(Ω)
$\Gamma \leftarrow \emptyset$ ▷ Γ stocks the set of users that start being served at slot t
$\overline{r} = 1$ ▷ $\overline{r}$, initialized by strategy 1, is the index of the strategy, under which users in Γ are served

10: **for** each packet $b \in \mathcal{B}(\Omega_t)$ **do**
11: $\quad \Gamma^* \leftarrow$ **UsersRequestPacket**(Ω_t, b)
12: $\quad r^* \leftarrow$ **LeastRobustStrategy**(Γ^*)
13: $\quad$ **if TotalReward**(Γ^*, r^*)>**TotalReward**$(\Gamma, \overline{r})$ **then**
14: $\quad\quad \Gamma \leftarrow \Gamma^*$
15: $\quad\quad \overline{r} = r^*$
16: $\quad$ **end if**
17: **end for**
18: $i \leftarrow$ **HighestPriorityRequest**(Γ)
19: **for** each user $k \in \Gamma$ **do**
20: $\quad t_k = t$, $r_k = \bar{r}$, $\beta_k = i$ $\quad \triangleright$ Start serving request k under strategy $\bar{r}$ and priority i at slot t
21: $\quad$ add (k, t_k, r_k, β_k) to $\mathcal{N}_t^*$
22: **end for**
23: **return** $\mathcal{N}_t^*$

Theorem 4.4. *If the slackness of each request is at least* $l_{max} + \min_{r \in \mathcal{R}_i} \tau_{r,b_i} - 1$*, i.e.,* $d_i - a_i + 1 \geq l_{max} + \min_{r \in \mathcal{R}_i} \tau_{r,b_i} - 1, \forall i \in \mathcal{N}$*, the competitive ratio of Algorithm 4 is upper-bounded by* B_{max}*, i.e., Algorithm 4 outputs a* $1/B_{max}-$*optimal feasible scheduling policy.*

Proof. Let $\mathcal{T}$ denote the set of slots, at each of which Algorithm 4 starts serving at least one request. We first prove that the utility of the optimal scheduling policy is upper-bounded by the total reward of the requests, each of which is ready at a slot in $\mathcal{T}$. For each slot $t \in \mathcal{T}$, we then derive the relationship between the total reward of the requests starting being served by Algorithm 4 at slot t and the total reward of the ready requests at slot t. Combining the above results allows us to establish $B_{max}-$competitive of Algorithm 4. □

Theorem 4.4 shows that the performance of our algorithm only scales with the largest $\mathcal{B}(\Omega_t)$, which only depends on the number of packets that are requested by the ready users at the current slot t. Therefore, the global efficiency of our algorithm does not degrade with the system size.

4.5 Numerical Analysis

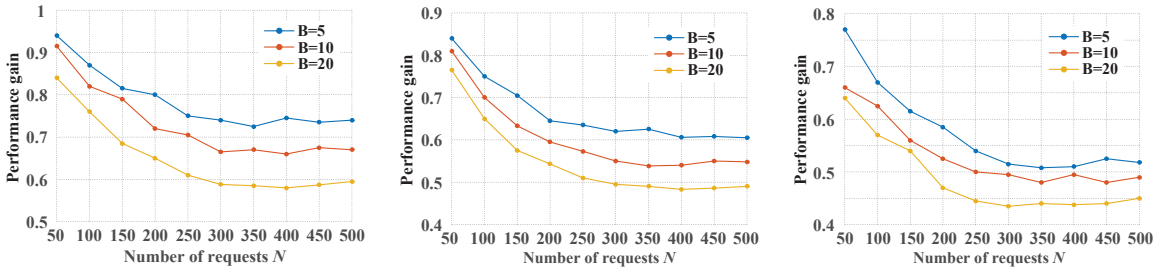
In this section, we conduct numerical analysis to evaluate the performance of the constant-factor offline and online scheduling approximation algorithms we develop. In our simulation, we trace the metric (3.8) to evaluate the performance of the optimum

scheduling policy compared to our algorithms. Specifically, we trace the maximal, average, and minimal values of Υ in our simulations.

In our simulations, the time horizon T is set to 200. For each strategy indexed by r, the number of time slots to transmit each packet is r under strategy r. We simulate three typical scenarios, in each of which we vary the number of requests N in the system from 50 to 500. For each N, we perform 50 simulation runs for each request parameter setting.

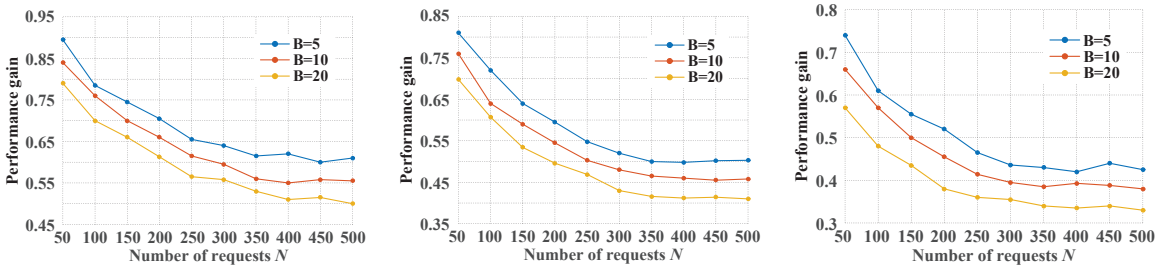
4.5.1 Scenario 1

In the first scenario, we randomly choose the parameters $a_i, d_i, b_i, \mathcal{R}_i$ such that $d_i - a_i + 1 \geq \min_{r \in \mathcal{R}_i} \tau_{r,b_i}$, and there are 5 strategies indexed from 1 to 5, i.e., $R = 5$. We run three experiments with $B = 5, 10$ and 20 respectively. The simulation results of the offline and online cases are illustrated in Figures 4.3 and 4.4.



(a) Maximum performance gains (b) Average performance gains (c) Minimum performance gains

Figure 4.3: Performance gains of Algorithm 3 for the first scenario in offline case



(a) Maximum performance gains (b) Average performance gains (c) Minimum performance gains

Figure 4.4: Performance gains of Algorithm 4 for the first scenario in online case

From the simulation results, we make the following observations.

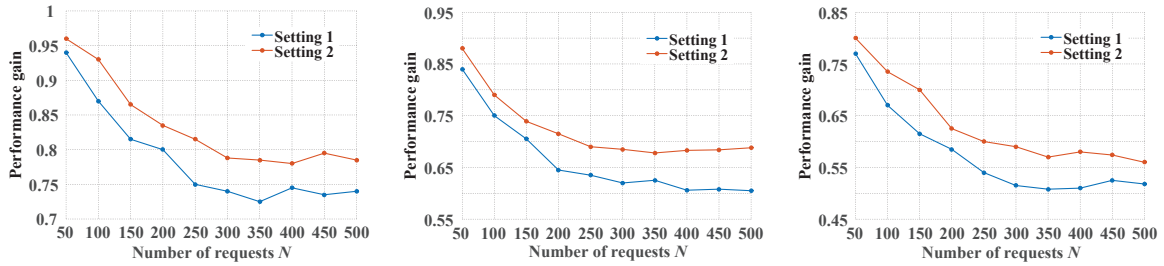
- Our algorithms achieve at least 43% of the optimal utility even in the worst case in the offline setting and at least 33% in the online setting, which demonstrate

the theoretical results we derive.

- Our offline algorithm performs better compared to our online algorithm, as the base station has the full knowledge of request information in the offline setting and naturally achieves better performance.
- When the number of requests N increases, the average performance gain first decreases in both offline and online settings, and then stabilize, indicating that the resource pool approaches its capacity limit.
- The smaller the number of packets there is, the better performance our algorithms achieve. This is because, when the number of packets is small, there are more opportunities for serving requests simultaneously, which potentially improves the system performance.

4.5.2 Scenario 2

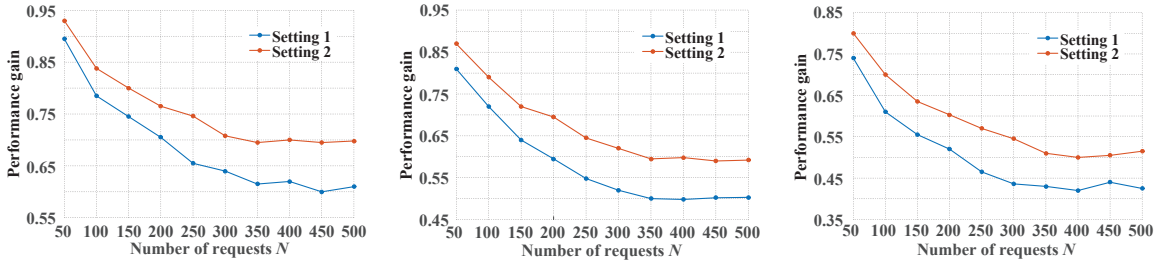
In this scenario, we set $B = 5$ and randomly choose the parameters $a_i, d_i, b_i, \mathcal{R}_i$, and there are 5 strategies indexed from 1 to 5, i.e., $R = 5$. We run the experiments in Scenario 2 including the following two settings: (1) $d_i - a_i + 1 \geq \min_{r \in \mathcal{R}_i} \tau_{r,b_i}$, and (2) $d_i - a_i + 1 \geq l_{max} + \min_{r \in \mathcal{R}_i} \tau_{r,b_i} - 1, \forall i \in \mathcal{N}$. The simulation results for this scenario are shown in Figures 4.5 and 4.6. From the results, we derive the following observations.



(a) Maximum performance gains (b) Average performance gains (c) Minimum performance gains

Figure 4.5: Performance gains of Algorithm 3 for Scenario 2 in offline case

- Our algorithms achieve at least 51% of the optimal utility in offline case, and 42% of the optimal utility in the online case.
- By comparing the performance gains between the two settings, as the slackness of each request is relaxed, our algorithms in the second setting perform better compared to the first setting.

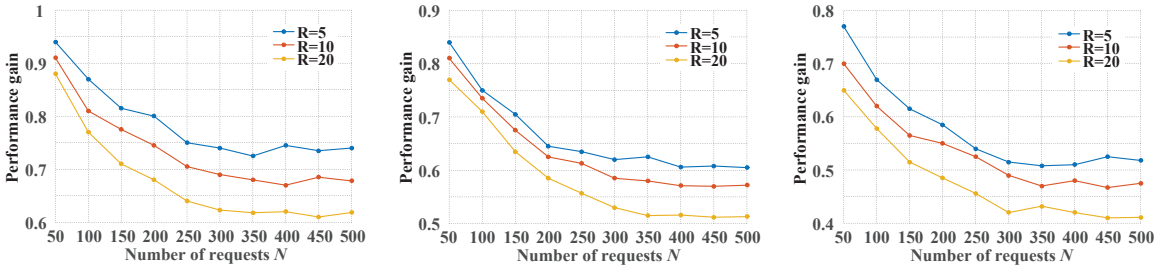


(a) Maximum performance gains (b) Average performance gains (c) Minimum performance gains

Figure 4.6: Performance gains of Algorithm 4 for Scenario 2 in online case

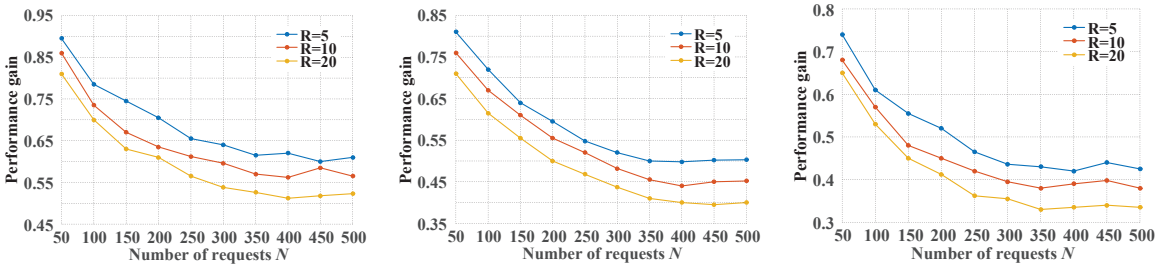
4.5.3 Scenario 3

In this scenario, we set $B = 5$ and randomly choose the parameters $a_i, d_i, b_i, \mathcal{R}_i$ such that $d_i - a_i + 1 \geq \min_{r \in \mathcal{R}_i} \tau_{r,b_i}, \forall i \in \mathcal{N}$. The experiments of Scenario 3 include the following three settings: (1) there are 5 strategies indexed from 1 to 5, i.e., $R = 5$, (2) there are 10 strategies indexed from 1 to 10, i.e., $R = 10$, and (3) there are 20 strategies indexed from 1 to 20, i.e., $R = 20$. The simulation results of this scenario are illustrated in Figures 4.7 and 4.8. From the results, we make the following observations.



(a) Maximum performance gains (b) Average performance gains (c) Minimum performance gains

Figure 4.7: Performance gains of Algorithm 3 for Scenario 3 in offline case



(a) Maximum performance gains (b) Average performance gains (c) Minimum performance gains

Figure 4.8: Performance gains of Algorithm 4 for Scenario 3 in online case

- Our algorithms achieve at least 41% of the optimal utility in offline case, and 33% of the optimal utility in the online case.
- As in the first scenario, the smaller the number of strategies there is, the better performance our algorithms achieve.

4.6 Conclusion

In this chapter, we have formulated and analyzed the downlink transmission scheduling problem with data sharing. We have studied both offline and online scheduling scenarios. In both cases, we have established the problem hardness and developed deterministic approximation algorithms with mathematically proven performance guarantee. We have complemented our theoretical analysis with numerical simulations that demonstrate the effectiveness of our algorithms in a variety of system settings.

4.7 Appendix

4.7.1 Integer Linear Problem Formulation of Offline Downlink Transmission Scheduling

The offline downlink transmission scheduling with data sharing problem **P3** can be formulated as a network utility maximization (NUM) problem as below.

$$\textbf{P3:} \quad \text{maximize} \sum_{1\le i\le N}\sum_{r\in\mathcal{R}_i} w_{i,r}x_{i,r}$$

subject to

$$t_i - t_j \ge \tau_{r_j,b_j}, \quad \forall i,j\in\mathcal{N}, t_i > t_j, r_i \ne r_j, x_{i,r_i} = x_{j,r_j} = 1. \tag{4.4a}$$

$$t_i - t_j \ge \tau_{r_j,b_j}, \quad \forall i,j\in\mathcal{N}, t_i > t_j, b_i \ne b_j, x_{i,r_i} = x_{j,r_j} = 1. \tag{4.4b}$$

$$t_j > t_i, \qquad \forall i,j\in\mathcal{N}, \beta_j > \beta_i, x_{i,r_i} = x_{j,r_j} = 1. \tag{4.4c}$$

$$\forall j\in\mathcal{N}, \beta_j = \beta_i, x_{j,r_j} = 1,$$

$$t_j = t_i, \qquad i = \min\{k : x_{k,r_k} = 1, j\ge k, k\in\mathcal{N}_{b_j}, \tag{4.4d}$$

$$r_k\in\mathcal{R}_j, a_j\le t_k\le d_j - \tau_{r_k,b_j}+1\}.$$

$$a_i \le t_i \le d_i - \tau_{r_i,b_i}+1, r_i\in\mathcal{R}_i, \quad x_{i,r_i} = 1, \forall i\in\mathcal{N}. \tag{4.4e}$$

$$x_{i,r_i}\in\{0,1\}, \quad \forall i\in\mathcal{N}, \forall r\in\mathcal{R}_i$$

where t_i is the time to start serving user i if its request is admitted, $x_{i,r}$ is the decision variable indicating whether to serve user i under strategy r, r_i and β_i are the transmission strategy and priority under which user i is served, respectively. Constraint (4.4a) implies that if any pair of requests is served under different strategies, the time intervals to serve them cannot overlap. Constraint (4.4b) implies that if any pair of served users requests different packets, the time intervals to serve them cannot overlap. Constraints (4.4c) and (4.4d) correspond to the adapted FIFO model. Constraint (4.4c) indicates that for any pair of request i and j, if request j is served under the lower priority than the priority under which request i is served, the time to start serving j cannot be earlier than the time to start serving i. Constraint (4.4d) implies that if request j is served, it must be served under the priority of itself or batched with a request with highest priority among the served users who request the same packet as j and whose time intervals can fit for serving j. Constraint (4.4e) indicates that if user i is served, i should be served under a strategy in $\mathcal{R}_i$ and there is enough time to transmit the requested packet to user i by its deadline.

4.7.2 Proof of Theorem 4.1

To prove its hardness, we consider the 0-1 Knapsack problem which is known to be NP-hard [26].

0-1 Knapsack problem. Given a positive capacity C and a set $\mathcal{N}$ of N elements indexed from 1 to N, each with a positive weight $c_i (i \in \{1, 2, \cdots, N\})$, find a set $\mathcal{S} \subseteq \{1, 2, \cdots, N\}$ such that $\sum_{i \in \mathcal{S}} c_i \leq C$ and $\sum_{i \in \mathcal{S}} c_i$ is maximized. The problem is known to be NP-hard [26].

We now show that the 0-1 Knapsack problem defined above, we construct an instance of **P3** as follows: there are N users indexed from $i = 1$ to N, each submitting a request $(0, T)$, i.e., each request needs the whole time horizon; there is only one transmission strategy that can cover all users in the base station, where each user requests a distinguished packet; the time to transmit the packet requested by each user i is c_i under the transmission strategy; the reward of each user i is c_i; as all requests are submitted at time 0, the service order is determined by user indexes. It is straightforward to see that a solution of the constructed instance of **P3** can be cast to a solution of the 0-1 Knapsack problem.

It then follows from the NP-hardness of the 0-1 Knapsack problem that **P3** is NP-hard.

In the disctete case where all the parameters and variables are restricted to integers, we can apply the same procedure to cast the Knapsack problem with integer weights to the discrete downlink transmission scheduling problem. It then follows from the NP-completeness of Knapsack problem with integer weights [23] that the discrete version of **P3** is also NP-complete.

4.7.3 Proof of Lemma 4.1

First, we prove that each IS of G maps to a feasible policy.

- The construction of type 1 edges ensures that the base station does not transmit multiple packets simultaneously.
- The construction of type 2 edges ensures that the priorities, under which the users are served, never violate our adapted FIFO model.
- The construction of type 3 edges ensures that for each user, at most one vertex covering it is chosen, i.e., each user is served at most once.

Therefore, each IS of G maps to a feasible policy.

We then prove the second property. Let $\mathcal{N}^*$ denote the set of users that are served by an optimal feasible policy. Following our notation, we use a quadruple (i, t_i, r_i, β_i) to denote a request i served in $\mathcal{N}^*$, where t_i denotes the starting time slot to serve request i, r_i denotes the strategy under which the request i is served, and β_i denotes the priority under which request i is served. We divide $\mathcal{N}^*$ into subsets such that the requests belonging to the same subset are batched under the same priority. Let $\mathcal{N}_0^*$ denote the set of batch headers. Clearly, each request in $\mathcal{N}^*$ belongs to a batch. It then follows from the optimality of $\mathcal{N}^*$ that

$$\mathcal{N}^* = \bigcup_{i \in \mathcal{N}_0^*} \mathcal{N}_{b_i,t_i}^+(i, r_i).$$

It follows from the feasible of $\mathcal{N}^*$ that any pair of requests $i, j \in \mathcal{N}_0^*$ with $i < j$ satisfies $t_i + \tau_{r_i,b_i} - 1 < t_j \leq d_j - \tau_{r_j,b_j} + 1$.

For each user i, a strategy r and a slot t, it follows from the definition of $\mathcal{N}_{b_i,t}^+(i, r)$ that the set of requests in $\mathcal{N}_{b_i,t}^+(i, r)$ includes all requests k with $k \geq i, b_k = b_i$ that fit into the time interval $[t, t + \tau_{r,b_i} - 1]$. Consider any packet b and any pair of users $i, j \in \mathcal{N}_0^*$ with $j < i$ requesting the packet b. We now prove that $r_j \in \mathcal{R} \setminus \mathcal{R}_i$ when there

exists a request in $\mathcal{N}^+_{b,t_i}(i, r_i)$ that can fit into the time interval $[t_j, t_j + \tau_{r_j,b} - 1]$, i.e., $\mathcal{N}^+_{b,t_j}(j, r_j) \cap \mathcal{N}^+_{b,t_i}(i, r_i) \neq \emptyset$. Assume by contradiction that $r_j \in \mathcal{R}_i$ when $\mathcal{N}^+_{b,t_j}(j, r_j) \cap \mathcal{N}^+_{b,t_i}(i, r_i) \neq \emptyset$. For each request $k \in \mathcal{N}^+_{b,t_j}(j, r_j) \cap \mathcal{N}^+_{b,t_i}(i, r_i)$, it holds that $i \leq k$ and $a_i \leq a_k \leq t_j$ because of the definitions of $\mathcal{N}^+_{b,t_i}(i, r_i)$ and $\mathcal{N}^+_{b,t_j}(j, r_j)$. It then follows from $r_j \in \mathcal{R}_i$ and $t_j + \tau_{r_j,b} - 1 < t_i < d_i$ that i fits into the time interval $[t_j, t_j + \tau_{r_j,b} - 1]$. Based on the adapted FIFO model, request i must be batched with j or another request with higher priority than j. It then indicates that i is not a batch header, i.e., $i \notin \mathcal{N}^*_0$, contradicting with $i \in \mathcal{N}^*_0$, where the proof is completed.

We then prove that there is a subset of vertices in $\mathcal{V}$, which is an IS, covering all users in $\mathcal{N}^*$. Consider each packet b. For each request $i \in \mathcal{N}^*_0$ requesting the packet b, it follows from the construction of G that we can find a vertex in $\mathcal{V}$, denoted by v^*_i, corresponding to the time interval $[t_i, t_i + \tau_{r_i,b_i} - 1]$, the strategy r_i, and the priority i. The requests covered by v^*_i is distinguished into two cases.

- If i has the highest priority among the users in $\mathcal{N}^*_0$ requesting packet b, i.e., $i = \min_{k \in \mathcal{N}^*_0, b_k = b} k$, then v^*_i covers requests $\mathcal{N}^+_{b,t_i}(i, r_i)$.
- Otherwise, we select the request $j = \max_{k \in \mathcal{N}^*_0, k < i, b_k = b} k$,
  - if there is no user in $\mathcal{N}^+_{b,t_i}(i, r_i)$ that can fit into the time interval $[t_j, t_j + \tau_{r_j,b_j} - 1]$, i.e., $\mathcal{N}^+_{b,t_i}(i, r_i) \cap \mathcal{N}^+_{b,t_j}(j, r_j) = \emptyset$, then v^*_i covers requests $\mathcal{N}^+_{b,t_i}(i, r_i)$.
  - if there exists a user in $\mathcal{N}^+_{b,t_i}(i, r_i)$ that can fit into the time interval $[t_j, t_j + \tau_{r_j,b_j} - 1]$, i.e., $\mathcal{N}^+_{b,t_i}(i, r_i) \cap \mathcal{N}^+_{b,t_j}(j, r_j) \neq \emptyset$, then v^*_i covers requests $\mathcal{N}_{b,t_i}(i, i^*; r_i, r_j)$, where $i^* = \max_{k \in \mathcal{N}^+_{b,t_i}(i,r_i) \cap \mathcal{N}^+_{b,t_j}(j,r_j)} k$. It holds that $r_j \in \mathcal{R} \setminus \mathcal{R}_i$.

We consider the set of vertices $\mathcal{V}^* = \{v^*_i\}_{i \in \mathcal{N}^*_0}$. It follows from the construction of vertices in $\mathcal{V}^*$ that all requests in $\mathcal{N}^*$ are covered by the vertices in $\mathcal{V}^*$, i.e., $\bigcup_{i \in \mathcal{N}^*_0} \mathcal{N}(v^*_i) = \bigcup_{i \in \mathcal{N}^*_0} \mathcal{N}^+_{b_i,t_i}(i, r_i) = \mathcal{N}^*$, where $\mathcal{N}(v)$ denotes the set of requests covered by v.

Next, we prove that the set of vertices in $\mathcal{V}^*$ is an IS of G. Consider any pair of vertices $u, v \in \mathcal{V}^*$ with $\beta_u < \beta_v$. Since for any pair requests $k_1, k_2 \in \mathcal{N}^*_0$ with $k_1 < k_2$ it holds that $t_{k_1} + \tau_{r_{k_1},b_{k_1}} - 1 < t_{k_2}$, the time intervals corresponding to u and v do not overlap, indicating that any pair of vertices in $\mathcal{V}^*$ is not connected by the type 1 edge. Based on the construction of the vertices in $\mathcal{V}^*$, we can derive that (1) $t_u < t_v$ and (2) $\mathcal{N}(u) \cap \mathcal{N}(v) = \emptyset$, indicating that any pair of vertices in $\mathcal{V}^*$ is not connected by the type 2 edge and type 3 edge.

By combining the above analysis, the set of vertices in $\mathcal{V}^*$ is an IS of G and covers all requests in $\mathcal{N}^*$, where the proof of the second property is completed.

4.7.4 Proof of Lemma 4.2

It follows from Algorithm 3 that the colors that cannot be used to v have been used to the neighbors of v when Algorithm 3 colors v. We first prove that the number of colors that have been used to the neighbors of v is upper-bounded by $(1+\varphi/l_{min})\hat{N} - 2\lfloor y_v^*\hat{N}\rfloor$ when Algorithm 3 colors v. The neighbors of v can be divided into the following three classes.

The first class of neighbors consist of the vertices that are connected with v by type 1 edges. When coloring the vertex v, all vertices u with $t_u < t_v$ have been colored and no vertex u with $t_u > t_v$ is colored because we color the vertices in $\mathcal{V}$ in non-decreasing order by their starting time. Thus, the colors, which cannot be used to v, have been used to color the first class of neighbors of v that are active at slot t_v. It follows from the constraint (4.2) that

$$\sum_{u\in\mathcal{V}, u \text{ is active at slot } t_v} \lfloor y_u^*\hat{N}\rfloor \leq \sum_{u\in\mathcal{V}, u \text{ is active at slot } t_v} y_u^*\hat{N} \leq \hat{N}.$$

Therefore, the number of colors that have been used to color the first class of neighbors of v is upper-bounded by $\hat{N} - \lfloor y_v^*\hat{N}\rfloor$ when Algorithm 3 colors v.

The second class of neighbors consist of the vertices that are connected with v by type-2 edges. For each user $k \in \mathcal{N}$, it follows from the constraints (4.3) that

$$\sum_{u\in\mathcal{V}^\alpha(k)} \lfloor y_u^*\hat{N}\rfloor \leq \sum_{u\in\mathcal{V}^\alpha(k)} y_u^*\hat{N} \leq \frac{\max_{j\in\mathcal{N}^\alpha(k)}(d_j - a_k + 1)}{l_{min}} \cdot \hat{N} \leq \frac{\varphi}{l_{min}} \cdot \hat{N},$$

where the last inequality follows from $\max_{j\in\mathcal{N}^\alpha(k)}(d_j - a_k + 1) \leq \varphi$ because of $a_j \leq a_k, \forall j \in \mathcal{N}^\alpha(k)$. Therefore, the colors that are used to the vertices in $\mathcal{V}^\alpha(k)$ is upper-bounded by $\frac{\varphi}{l_{min}} \cdot \hat{N}$.

For each request k, we assume that the vertices in $\mathcal{V}^\alpha(k)$ are adjacent to each other in the current proof. Clearly, the edges connecting each pair of vertices in $\mathcal{V}^\alpha(k)$ include all type-2 edges between the vertices in $\mathcal{V}^\alpha(k)$.

Based on the construction of type-2 edges, there is no type-2 edge between any vertex in $\mathcal{V}^\alpha(k_1) \setminus \mathcal{V}^\alpha(k_2)$ and any vertex in $\mathcal{V}^\alpha(k_2) \setminus \mathcal{V}^\alpha(k_1)$. When Algorithm 3 colors

the vertices in $\mathcal{V}^\alpha(k_2)$, all vertices $u \in \mathcal{V}^\alpha(k_1) \setminus \mathcal{V}^\alpha(k_2)$ have been colored because of $t_u < \min\{t_\nu : \nu \in \mathcal{V}^\alpha(k_2)\}$. Therefore, the colors that are used to the vertices in $\mathcal{V}^\alpha(k_1) \setminus \mathcal{V}^\alpha(k_2)$ can be used to the vertices in $\mathcal{V}^\alpha(k_2) \setminus \mathcal{V}^\alpha(k_1)$ if there is also no type 1 and 3 edge. Hence, the number of colors that have been used to color the second class of neighbors is upper-bounded by $\frac{\varphi}{l_{min}} \cdot \hat{N} - \lfloor y_v^* \hat{N} \rfloor$ when Algorithm 3 colors v.

The third class of neighbors consist of the vertices that are connected with v by type-3 edges. Consider each request k covered by v. For each vertex $u \in \mathcal{V}_k$, the priority of u is higher than or equal to the priority of k, and the vertices in $\mathcal{V}_k$ are valid at slot a_k. It then follows from the definition of $\mathcal{V}^\alpha(k)$ that the vertices in $\mathcal{V}_k$ are included by $\mathcal{V}^\alpha(k)$. We derive the number of colors that are occupied by the second class of neighbors based on that, for each request $j \in \mathcal{N}$, the vertices in $\mathcal{V}^\alpha(j)$ are adjacent to each other. Therefore, the colors that have been used to the third class of neighbors are included by the colors already used to the second class of neighbors.

By combining the above analysis, the number of colors that have been used to the neighbors of v is upper-bounded by $(1+\varphi/l_{min})\hat{N} - 2\lfloor y_v^* \hat{N} \rfloor$. As $|\mathcal{C}| = (1+\varphi/l_{min})\hat{N} - 1$ and $\lfloor y_v^* \hat{N} \rfloor \geq 1$, there are at least $\lfloor y_v^* \hat{N} \rfloor$ available colors in $\mathcal{C}$ when Algorithm 3 colors v. Thus, v is colored by $\lfloor y_v^* \hat{N} \rfloor$ colors. The lemma is thus proved.

4.7.5 Proof of Theorem 4.2

Denote the utility of the optimal solution of **P3** by OPT. Clearly, the set of vertices sharing a same color is an IS of G. It follows from Lemma 4.1 that each IS of G maps to a feasible scheduling policy. It follows from Lemma 4.2 and $|\mathcal{C}| \leq (1+\varphi/l_{min})\hat{N} - 1$ that Algorithm 3 outputs a coloring of G using at most $(1+\varphi/l_{min})\hat{N} - 1$ colors. We thus get at most $(1+\varphi/l_{min})\hat{N} - 1$ ISes, denoted by $\mathcal{I}_i, 1 \leq i \leq I$, where $I \leq (1+\varphi/l_{min})\hat{N} - 1$. Let $\mathcal{I}$ denote the IS output by Algorithm 3. Since each vertex $v \in \mathcal{V}$ receives $\lfloor \hat{N} y_v^* \rfloor$ colors, we have

$$\begin{aligned}
\sum_{v \in \mathcal{I}^*} w_v &\geq \frac{\hat{N}}{|\mathcal{C}|} \left(\sum_{v \in \mathcal{V}} w_v y_v^* - \sum_{v \in \mathcal{V}} \frac{w_v}{\hat{N}} \right) \\
&> \frac{1}{1+\varphi/l_{min}} \left(OPT - \frac{RM\delta}{RM\delta^{1+\epsilon}} \max_{v \in \mathcal{V}} w_v \right) \\
&= \frac{1}{1+\varphi/l_{min}} \left(OPT - \frac{1}{\delta^\epsilon} \max_{v \in \mathcal{V}} w_v \right).
\end{aligned}$$

where the second inequality follows that the value of an optimal fractional LP solution is an upper bound on the value of an optimal feasible scheduling policy.

As it holds that $\max_{v\in\mathcal{V}} w_v \leq OPT$, we then have

$$\sum_{v\in\mathcal{I}^*} w_v > \frac{1}{1+\varphi/l_{min}}(OPT - \frac{1}{\delta^\epsilon}\cdot OPT).$$

Noticing that asymptotically δ is sufficiently large, Theorem 4.2 is thus proved.

4.7.6 Proof of Theorem 4.4

Let $\mathcal{O}$ denote the set of requests that are served by the optimum scheduling policy. We now prove that each request in $\mathcal{O}$ is ready at a slot in $\mathcal{T}$. Consider each request $k \in \mathcal{O}$. If $a_k \in \mathcal{T}$, clearly, the priorities to serve requests that start being served before slot a_k is higher than the priority of request k. Therefore, request k is ready at slot a_k, i.e., $k \in \Omega_{a_k}$; if $a_k \notin \mathcal{T}$, let $t = \max\{t' : t' \in \mathcal{T}, t' < a_k\}$, and let τ^* denote the number of slots to serve the users in $\mathcal{N}_t^*$. We prove that request k is ready at slot $t+\tau^*$ and $t+\tau^* \in \mathcal{T}$. We prove it in the following three steps.

- **Step 1:** We prove that $t < a_k < t+\tau^*$. Assume by contradiction that $a_k \geq t+\tau^*$. It follows from $t = \max\{t' : t' \in \mathcal{T}, t' < a_k\}$ and $a_k \notin \mathcal{T}$ that each slot in $[t+\tau^*, a_k]$ is free when Algorithm 4 considers it. Because request k is ready at slot a_k, Algorithm 4 starts serving at least one request at slot a_k, i.e., $a_k \in \mathcal{T}$, contradicting with $a_k \notin \mathcal{T}$. It then holds that $t < a_k < t+\tau^*$.

- **Step 2:** We prove that request k is ready at slot $t+\tau^*$. Since the slackness of each request i is at least $l_{max} + \min_{r\in\mathcal{R}_i} \tau_{r,b_i} - 1$, we can derive that $a_k < t+\tau^* \leq d_k - \min_{r\in\mathcal{R}_i} \tau_{r,b_i} + 1$ because of $t < a_k < t+\tau^*$ and $\tau^* \leq l_{max}$. It also holds that starting serving request k at slot $t+\tau^*$ does not violate the FIFO model because of $a_j \leq t < a_k$ and $j < k$ for any j with $t_j < t+\tau^*$. Therefore, request k is ready at slot $t+\tau^*$, i.e., $k \in \Omega_{t+\tau^*}$.

- **Step 3:** We prove that $t+\tau^* \in \mathcal{T}$. Because requests in $\mathcal{N}_t^*$ are served completely at slot $t+\tau^*-1$, slot $t+\tau^*$ is free when Algorithm 4 considers it. Therefore, Algorithm 4 starts serving at least one request at slot $t+\tau^*$ since k is ready at slot $t+\tau^*$, i.e., $t+\tau^* \in \mathcal{T}$.

Based on the above analysis, the total reward of the optimum scheduling policy can be upper-bounded by $\sum_{t\in\mathcal{T}}\sum_{k\in\Omega_t} w_k$, i.e., $\sum_{k\in\mathcal{O}} w_k \leq \sum_{t\in\mathcal{T}}\sum_{k\in\Omega_t} w_k$.

Consider each slot $t \in \mathcal{T}$. Because Algorithm 4 starts serving a set of users in Ω_t, which request the same packet, with the maximum total reward among the packets in

$\mathcal{B}(\Omega_t)$, the total reward of requests in Ω_t is

$$\sum_{k \in \Omega_t} w_k \leq |\mathcal{B}(\Omega_t)| \sum_{k \in \mathcal{N}_t^*} w_k$$

Therefore, the total reward of the optimum scheduling policy can be upper-bounded by

$$\sum_{k \in \mathcal{O}} w_k \leq \sum_{t \in \mathcal{T}} \sum_{k \in \Omega_t} w_k \leq \sum_{t \in \mathcal{T}} |\mathcal{B}(\Omega_t)| \sum_{k \in \mathcal{N}_t^*} w_k \leq B_{max} \cdot \sum_{t \in \mathcal{T}} \sum_{k \in \mathcal{N}_t^*} w_k.$$

Theorem 4.4 is thus proved.

Chapter 5

Contiguous-resource Batching Task Scheduling

This chapter investigates a two-dimensional task scheduling problem, where a set of tasks need to be executed on a pool of continuous resource, each requiring a certain amount of time and contiguous resource; some tasks can be executed simultaneously in batch by sharing the resource, while others requiring exclusive use of the resource; tasks are served in the FIFO manner. We seek an optimal resource allocation and the related scheduling policy maximizing the overall system utility. The above problem is termed as *contiguous-resource batching task scheduling problem*. The scheduling problems addressed in the previous two chapters can be regarded as the degenerated instances of the contiguous-resource batching task scheduling problem, where tasks require the entire resource in one of the dimension.

Compared with existing works on bandwidth and storage allocation problem in Section 2.4, and task scheduling problem in Section 2.2, our contiguous-resource batching task scheduling problem we address has the following constraints that has not been holistically addressed before: (1) the allocated resource for each request must be contiguous, (2) the requests are divided into non-overlapping groups, with those in the same group executable in batched, (3) the requests should be served by the adapted FIFO model. We note that it is the combination of these constraints that makes our problem non-trivial and requires a systematical treatment.

In this chapter, we investigate both offline and online scheduling settings. In both cases, we first establish the problem hardness and then develop approximation algorithms with proven performance guarantee. We further complement our theoretical analysis with numerical simulations that demonstrate the effectiveness of our algorithms

in a variety of system settings.

5.1 Introduction

As an instantiation of our work, we consider the channel bonding problem in wireless networks. The proliferation of wireless mobile networks and the ever-increasing density of wireless devices underscore the necessity for efficient allocation and sharing of the radio spectrum resource. *Spectrum bonding*, or channel bonding, is widely regarded as an effective enabling technique that combines *contiguous* spectrum fragments to create a wideband channel for data transmission, thus significantly increasing the spectrum efficiency.

In this chapter, we focus on the design and analysis of spectrum bonding and the related transmission scheduling algorithms in dynamic spectrum access systems. Specifically, we study a generic scenario where a number of users can access a continuous frequency band; each user has a communication task and thus issues a request composed of the amount of contiguous bandwidth needed to accomplish the task and the duration of the task; admitted requests are served according to the FIFO model; non-interfering users may be allocated overlapping spectrum bands simultaneously. We are interested in designing spectrum bonding and the related scheduling algorithms maximizing the overall system utility.

The above algorithmic problem also arises in the context of dynamic storage allocation in computer systems, where a pool of continuous memory space is shared among a number of processes, each requesting to access a certain amount of contiguous memory for a certain amount of time; a request can be of type either *read* or *write*; multiple read requests may be served simultaneously, while a write request requires exclusive use of the requested memory; admitted requests are served according to the FIFO model. The central problem is to design memory allocation algorithms maximizing the aggregated system utility, e.g., maximizing the number of admitted requests.

The above two examples further push us to formulate a generic task scheduling problem, where a set of independent tasks need to be executed on a pool of continuous resource, each requiring a certain amount of time and certain amount of contiguous resource. Some tasks can be executed simultaneously in batch by sharing the resource, while others requiring exclusive use of the resource. Tasks are served in the FIFO manner. We seek an optimal resource allocation and scheduling policy maximizing the overall system utility.

In this chapter, we embark on an algorithmic study of the above contiguous-resource batching task scheduling problem by instantiating our analysis in the context of spectrum bonding for the sake of concreteness. We investigate both offline and online scheduling settings. In both cases, we develop approximation algorithms with proven performance guarantee in terms of approximation and competitive ratios, respectively. Technically, we first demonstrate the hardness of our problem by showing that (1) the offline problem is NP-hard, (2) the online problem in its generic form cannot be approximated with any finite competitive ratio. Given the hardness of both the offline and online problems, we then seek approximation algorithms. In the offline problem, we develop an algorithmic framework by mapping the problem to a particular variant of the MWIS problem, termed as *maximum weighted regular independent set* (MWRIS), by integrating the specific constraints posed by our problem; we then solve the LP relaxation of MWRIS problem; we further develop a coloring-based algorithm that rounds the solution of the LP relaxation to an integer solution and establish its performance bound. In the online problem, we develop an iterative algorithm that greedily selects the most profitable tasks and prove that, as long as each task has certain slackness (cf. Section 5.4 for details), our algorithm can achieve finite competitive ratio.

The rest of this chapter is organized as follows. Section 5.2 formulates the contiguous-resource batching task scheduling problem. Section 5.3 analyzes the offline setting and develops the scheduling algorithmic framework. Section 5.4 studies the online setting and presents the design of an online scheduling algorithm. Section 5.5 presents the simulation results demonstrating the efficiency of our proposed algorithms. Section 5.6 concludes the chapter.

5.2 System Model

For the sake of concreteness, we present the system model of our problem in the context of channel bonding. Nevertheless, as explained in the Introduction, the model can be readily applied in a variety of task scheduling scenarios where a pool of resource needs to be allocated to a set of *tasks* following the FIFO model, some of which can be served together on common resource. Therefore, the following description and the use of terms such as *frequency band*, *user*, *transmission*, etc., should be understood generically.

We consider a dynamic spectrum access system where a set $\mathcal{N}$ of N users can access a common frequency band. We focus on the discrete case where both time and frequency

bands are discretized. To make our analysis concise without losing generality, we assume that time is divided to unit-length slots from 0 to T and the available frequency band, aka *white space*, is normalized to $[0, F]$ with the finest frequency granularity being 1. Each user i submits its bandwidth request in the form of a quadruple (a_i, d_i, f_i, l_i), where a_i denotes the arrival time of the request at the spectrum broker that manages the spectrum resource, d_i denotes the deadline before which the request needs to be satisfied if admitted by the broker, f_i is the quantity of contiguous frequency band requested, l_i is the number of slots the user requests to use the spectrum, $(d_i - a_i + 1)$ is the *slackness* of the user i. In other words, if the spectrum broker decides to serve request i, it needs to reserve f_i amount of contiguous frequency band to user i for l_i contiguous slots in the time interval $[a_i, d_i]$, as illustrated in Figure 5.1. We associate a reward w_i for each request i.

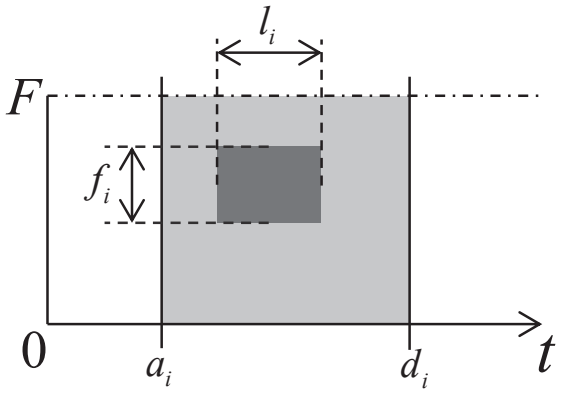


Figure 5.1: Illustration of an admitted request i

A particularity of radio resource is its reusability, meaning that multiple non-interfering users can use the same spectrum simultaneously. Formally, we consider a generic scenario where users are divided into B $(1 \leq B \leq N)$ groups, and users in the same group can be served simultaneously. We use b_i $(1 \leq b_i \leq B)$ to denote the group to which request i belongs. For any pair of requests i and j, request j can reuse the spectrum of i if and only if $b_i = b_j$. Let $\mathcal{N}_b$ denote the set of requests in $\mathcal{N}$ belonging to group b. Table 5.1 lists the main notations used in the chapter.

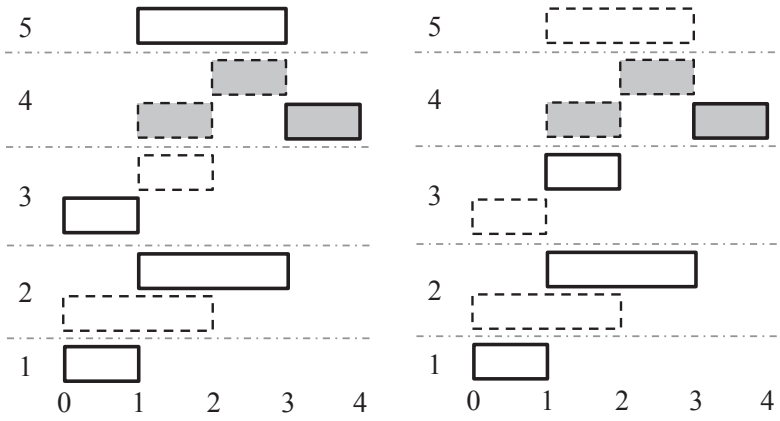
In the above generic model, we are interested in both offline and online optimal spectrum bonding and scheduling policies for the spectrum broker to maximize the total reward. Both policies follow the FIFO service model such that admitted requests are queued at the broker in the same order they arrive and a request is cleared when the corresponding user is served. Without loss of generality, we assume that $a_i \leq a_j$ for any $1 \leq i < j \leq N$; in case where $a_i = a_j$ and $i < j$, request i should be served first.

Table 5.1: Main notations

$\mathcal{N}$	request set
N	number of requests, $N = \lvert\mathcal{N}\rvert$
T	time horizon in number of time slots
F	frequency band in number of frequency granularity
B	number of groups
a_i	arriving time of request i
d_i	deadline of request i
f_i	quantity of contiguous frequency band requested by i
l_i	number of slots user i requests to use the spectrum
b_i	group index of request i
w_i	reward of request i
G	request graph
$\mathcal{V}$	set of vertices in G
$\mathcal{E}$	set of edges in G
$\mathcal{V}_i^+$	set of the elementary vertices of request i
$\mathcal{V}_i$	set of the elementary vertices of request i plus the induced vertices assigned to i
t_v	left boundary (i.e., starting time) of the rectangle corresponding to vertex v
s_v	lower boundary (i.e., starting spectrum) of the rectangle corresponding to vertex v
l_v	length of the rectangle corresponding to vertex v
h_v	height of the rectangle corresponding to vertex v
w_v	weight of vertex v
β_v	priority of v
β_i	priority under which request i is served
$\mathcal{N}_v^+$	set of requests k with lower priority than i that can be served together with i, for each elementary vertex $v \in \mathcal{V}_i^+$
$\mathcal{N}_b$	set of requests in $\mathcal{N}$ belonging to group b
F_b	$\max_{k\in\mathcal{N}_b} f_k$
f_b	$\min_{k\in\mathcal{N}_b} f_k$
L_b	$\max_{k\in\mathcal{N}_b} l_k$
l_b	$\min_{k\in\mathcal{N}_b} l_k$
f_{min}	$\min_{k\in\mathcal{N}} f_k$
f_{max}	$\max_{k\in\mathcal{N}} f_k$
l_{min}	$\min_{k\in\mathcal{N}} l_k$
l_{max}	$\max_{k\in\mathcal{N}} l_k$
R	$\max_{1\le b\le B}(F_b - f_b + 1)(L_b - l_b + 1)$
M	$\max_{1\le b\le B} \lvert\mathcal{N}_b\rvert$
δ	$\sum_{i\in\mathcal{N}}(d_i - a_i - l_i + 2)$
y_v	binary variable indicating whether or not vertex v is selected in the IS

In our problem, the standard FIFO model needs to be adapted as below to take into account spectrum reuse.

Definition 5.1 (Adapted FIFO model)**.** *Any scheduling policy satisfying the following requirement is called an adapted FIFO model. When serving request i, any request $j > i$ that can reuse the spectrum of i must also be served simultaneously with request i, as long as j can fit in the service time of i and the spectrum allocated to i.*



(a) Our FIFO model (b) Standard FIFO model

Figure 5.2: Illustration of our adapted FIFO model vs. standard FIFO model.

Example 5.1. *To illustrate the adapted FIFO model against the standard FIFO model, we consider an example composed of five requests, indexed from* 1 *to* 5*, with the corresponding request quadruples* $(0, 0, F, 1)$*,* $(0, 2, F, 2)$*,* $(0, 1, F, 1)$*,* $(1, 3, F, 1)$ *and* $(1, 2, F, 2)$*, respectively. Requests* 1*,* 2*,* 3 *and* 5 *belong to group* 1*, request* 4 *belongs to group* 2*. The reward of each request is* 1*. The rectangles in Figure 5.2 illustrate the possible executions of the requests, with the height being the quantity of requested spectrum and the length the requested execution time.*

- *Figure 5.2(a) illustrates the optimal scheduling policy under the adapted FIFO model, which serves requests* 1 *and* 3 *in slot* 0*, requests* 2 *and* 5 *in slot* 1*, request* 4 *in slot* 3*. All the five requests are served.*

- *Figure 5.2(b) illustrates the optimal scheduling policy under the standard FIFO model, which serves request* 1 *in slot* 0*, requests* 2 *and* 3 *in slot* 1*, request* 4 *in slot* 3*, and rejects request* 5*, or request* 1 *in slot* 0*, requests* 2*,* 3 *and* 5 *in slot* 1*, and rejects request* 4*. Different from our model, a request, if admitted, should*

be started no later than any admitted request arriving later. We can check that under the standard FIFO model, at most four requests can be served.

We remark that (1) the adapted FIFO model takes into account the spectrum reuse and is thus more flexible and naturally leads to better efficiency, as demonstrated by the illustration example, (2) it is technically more involved than the standard FIFO model.

In the adapted FIFO model, for a number of requests served simultaneously, among which the highest priority request is i, we say that these requests are served, or *batched*, under the priority i by reusing the spectrum allocated to i, and we call i the *head* of the batched requests.

We conclude this section by defining the *feasible scheduling policy* for contiguous-resource batching task scheduling.

Definition 5.2 (Feasible Scheduling Policy for Contiguous-resource Batching Task Scheduling)**.** *We call a scheduling policy feasible if*

- *it is an adapted FIFO model;*
- *each request is served at most once;*
- *the time-frequency blocks allocated to any pair of requests from different groups do not overlap.*

5.3 The Offline Case

5.3.1 Problem Formulation and Hardness

In this section, we consider the offline channel bonding problem, denoted by **P4**. In the following theorem, we prove that **P4** is NP-hard. To make our presentation more streamlined, we provide integer linear programming formulation of **P4** and the proof sketch of the lemmas and theorems in the main text and defer the detailed proof in Appendix 5.7.

Theorem 5.1 (Hardness of **P4**)**.** ***P4*** *is NP-hard. Its discrete version is NP-complete.*

Proof Sketch. We prove that the classical 0-1 Knapsack problem can be reduced to **P4**. It then follows from the NP-hardness of the 0-1 Knapsack problem that **P4** is NP-hard. It follows from NP-completeness of Knapsack problem with integer weights that the discrete version of **P4** is also NP-complete. □

Given the NP-hardness of **P4**, we devote our efforts to designing approximation algorithms. Our central idea is to construct a graph, termed as *request graph*, to capture the relationships among requests, given spectrum reusability and the constraint imposed by the adapted FIFO model. We then cast **P4** to the MWIS problem in the constructed request graph.

Conceptually, we can instantiate **P4** on an Euclidean plan with the x-axis being time and the y-axis being frequency. Each request i can then be cast to an axis-aligned rectangle of height f_i, length l_i whose reward is w_i. In order to be served, the above rectangular needs to be located within the rectangle $[a_i, d_i] \times [0, F]$. **P4** thus maps to the problem of finding a subset of rectangles corresponding to different requests maximizing the total reward, by taking into account the adapted FIFO model and spectrum reusability.

The main part of this section is organized in five subsections. We first construct the request graph and cast **P4** to the problem of MWIS. We then explore the structural properties of the constructed request graph to formulate the LP relaxation of **P4**. We further develop an approximation algorithm based on rounding the solution of the LP relaxation to an integer solution. The approximation ratio of our algorithm is established to complete the section.

5.3.2 Request Graph

The request graph, denoted by $G \triangleq (\mathcal{V}, \mathcal{E})$, consists of the following vertices and edges.

5.3.2.1 Vertices

We construct the following two types of vertices.

Elementary vertices. For each request $i \in \mathcal{N}$, we create a vertex for each rectangle of height f_i and length l_i in the rectangle $[a_i, d_i] \times [0, F]$. We call these vertices the elementary vertices of i. Let t_v and s_v denote the left boundary (i.e., the starting time) and the lower boundary (i.e., the starting spectrum) of the rectangle corresponding to v. Let l_v and h_v denote the length and height of the rectangle corresponding to v. Each elementary vertex v of i corresponds to the execution of i in the two-dimensional time-frequency block $[t_v, t_v + l_i - 1] \times [s_v, s_v + f_i - 1]$. If the scheduler decides to execute i using the time-frequency block corresponding to v, we say that i is *instantiated* by v. To take into account the adapted FIFO model, each elementary vertex of request i

is assigned a priority level i. For each elementary vertex v of request i, let $\mathcal{N}_v^+$ denote the set of requests $k > i$, i.e., with lower priority, that can be served together with i by reusing the time-frequency block of v, i.e.,

$$\mathcal{N}_v^+ \triangleq \{k : k > i, k \in \mathcal{N}_{b_i}, l_k \leq l_i, f_k \leq f_i, t_v + l_i - l_k \geq a_k, d_k - t_v + 1 \geq l_k\}.$$

Induced vertices. Consider each elementary vertex v of each request i. For each request $j \in \mathcal{N}_v^+$, we create an induced vertex u for each rectangle of height f_j and length l_j in the rectangle $[s_v, s_v + f_i - 1] \times [\max\{t_v, a_j\}, \max\{t_v, a_j\} + l_i - 1]$, and assign u to j. We set the priority of u to i. We say that u reuses the time-frequency block corresponding to v. We define u as a *child* of v, and inversely v as the *parent* of u, denoted by $v \prec u$. We note that an elementary vertex may have multiple children, while an induced vertex has only one parent.

We denote $\mathcal{V}_i$ the set of the elementary vertices of request i plus the induced vertices assigned to i. For each vertex $v \in \mathcal{V}_i$, we say that v covers i. We define a weight w_v for v and set $w_v = w_i$. Let β_v denote the priority of vertex v. Mathematically, a smaller β indicates a higher priority.

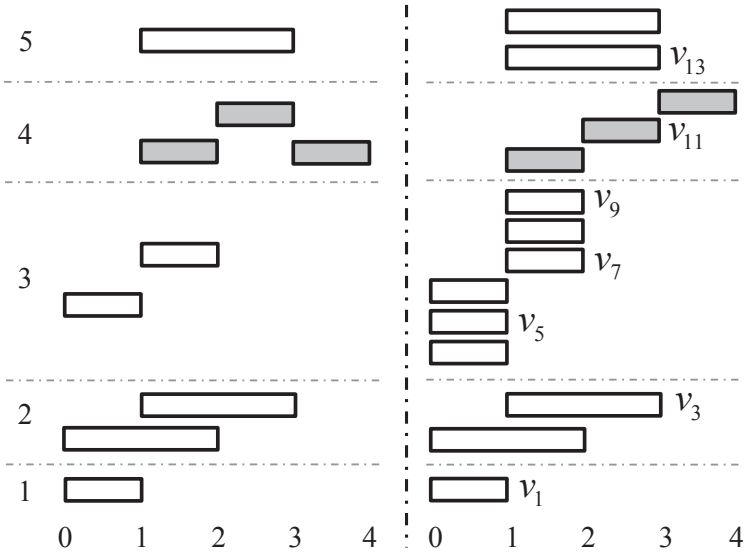


Figure 5.3: Illustration of vertex construction for Example 5.2

Example 5.2. *Figure 5.3 illustrates the vertex construction for the example in Figure 5.2(a). The left subfigure replots the possible cases to serve each request independently. The right subfigure plots the created vertices, further listed in Table 5.2. The unique optimal scheduling policy, starting serving requests* 1 *and* 3 *under the priority*

of 1 *in slot* 0*, requests* 2 *and* 5 *under the priority of* 2 *in slot* 1*, request* 4 *under the priority of* 4 *in slot* 3*, maps to vertices* v_1*,* v_4*,* v_3*,* v_{13}*, and* v_{12}*.*

We now analyze the complexity of creating the vertices in G. Denote $R \triangleq \max_{1\le b\le B}(F_b - f_b + 1)(L_b - l_b + 1)$, where $F_b \triangleq \max_{k\in\mathcal{N}_b} f_k$, $f_b \triangleq \min_{k\in\mathcal{N}_b} f_k$, $L_b \triangleq \max_{k\in\mathcal{N}_b} l_k$, $l_b \triangleq \min_{k\in\mathcal{N}_b} l_k$. Let $f_{min} \triangleq \min_{i\in\mathcal{N}} f_i$, $M \triangleq \max_{1\le b\le B} |\mathcal{N}_b|$, and $\delta \triangleq \sum_{i\in\mathcal{N}}(d_i - a_i - l_i + 2)$. There are $(F - f_i + 1)(d_i - a_i - l_i + 2)$ elementary vertices for each request i. The total number of elementary vertices sums up to $O((F - f_{min})\delta)$. For each elementary vertex v, there are $\sum_{j\in\mathcal{N}_v^+}(l_v - l_j + 1)(h_v - f_j + 1)$ induced vertices reusing the time-frequency block of v, and the complexity for listing the requests in $\mathcal{N}_v^+$ is $O(M)$. Hence, the total number of vertices sums to $O(RM(F - f_{min})\delta)$, and the construction of the vertices in G takes $O(RM(F - f_{min})\delta)$, asymptotically $O(RMF\delta)$, time.

Table 5.2: Constructed vertices in Example 5.2

Vertex v	Covered user	Priority	Rectangle	Parent
v_1	1	1	$[0,1)\times[0,F]$	
v_2	2	2	$[0,2)\times[0,F]$	
v_3	2	2	$[1,3)\times[0,F]$	
v_4	3	1	$[0,1)\times[0,F]$	v_1
v_5	3	2	$[0,1)\times[0,F]$	v_2
v_6	3	3	$[0,1)\times[0,F]$	
v_7	3	2	$[1,2)\times[0,F]$	v_2
v_8	3	2	$[1,2)\times[0,F]$	v_3
v_9	3	3	$[1,2)\times[0,F]$	
v_{10}	4	4	$[1,2)\times[0,F]$	
v_{11}	4	4	$[2,3)\times[0,F]$	
v_{12}	4	4	$[3,4)\times[0,F]$	
v_{13}	5	2	$[1,3)\times[0,F]$	v_3
v_{14}	5	5	$[1,3)\times[0,F]$	

5.3.2.2 Edges

We construct the following three types of edges.

Type-1: inter-user edges characterizing interference among requests belonging to different groups. For each pair of elementary vertices $v \in \mathcal{V}_i^+$ and $u \in \mathcal{V}_j^+$ with $b_i \neq b_j$, we construct an edge between v and u if their corresponding rectangles overlap, making it impossible to instantiate i by v and also j by u.

Type-2: inter-user edges modeling adapted FIFO model. This type is further classified into two cases.

- For each pair of elementary vertices $v \in \mathcal{V}_i^+$ and $u \in \mathcal{V}_j^+$, we construct an edge between v and u if $i > j$ and $t_v < t_u$ or $i < j$ and $t_v > t_u$, indicating that their corresponding requests cannot be both instantiated by u and v under the adapted FIFO model.

- For any pair of elementary vertices $v \in \mathcal{V}_i^+$ and $u \in \mathcal{V}_j^+$ satisfying $j \in \mathcal{N}_v^+$ and $t_v \leq t_u$, we construct an edge between them. The implication of this class of edges is below. Suppose request i is instantiated by v. It follows from $j \in \mathcal{N}_v^+$ that request j can fit into the service time of i and the spectrum allocated to i. It then follows from the adapted FIFO model that if request j is served, it must be batched with i or another request with higher priority than i. Hence, i and j cannot be both instantiated by v and u.

Type-3: intra-user edges. For each request $i \in \mathcal{N}$, we construct an edge between each pair of vertices in $\mathcal{V}_i$. The intra-user edges model the constraint that any user is served at most once.

We now analyze the complexity of creating the edges in G. As the number of elementary vertices in G is $O((F - f_{min})\delta)$, the construction of type 1 and 2 edges can be completed in $O((F - f_{min})^2\delta^2)$, asymptotically $O(F^2\delta^2)$, time. For each request i, there are at most $RM(F - f_i + 1)\delta_i$ vertices in $\mathcal{V}_i$. Therefore, the construction of type 3 edges takes $O\left(\sum_{i\in\mathcal{N}} \delta_i^2(F - f_{min})^2R^2M^2 + F^2\delta^2\right)$, asymptotically $O\left(F^2R^2M^2\sum_{i\in\mathcal{N}} \delta_i^2\right)$, time.

Example 5.3. *Figure 5.4 illustrates the type-1 and type-2 edges in G corresponding to Example 5.2 with the vertices listed in Table 5.2. Gray vertices are elementary vertices; solid edges correspond to type-1 edges; dashed edges and grey edges correspond to the two cases of type-2 edges.*

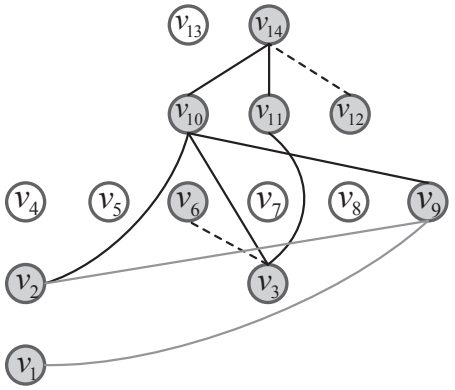


Figure 5.4: Illustration of type-1 and type-2 edges in G

5.3.3 From Channel Bonding to Maximum Weighted Regular Independent Set

Armed with the constructed request graph G, we can cast **P4** to the MWIS problem, more precisely, a variant of the MWIS problem called maximum weighted regular independent set problem. We start by defining the *regularly independent set* (RIS), which can be regarded as an enhanced version of IS adapted to our problem with frequency reuse. By choosing an IS in G we mean to serve the requests instantiated by the vertices in the chosen IS.

Definition 5.3 (Regular Independent Set)**.** *Given the request graph G, a vertex set $\hat{\mathcal{V}} \subseteq \mathcal{V}$ is a regular independent set if it satisfies the following two properties.*

- *Any pair of vertices in $\hat{\mathcal{V}}$ is not adjacent to each other.*
- *The parent of any induced vertex in $\hat{\mathcal{V}}$ is also in $\hat{\mathcal{V}}$.*

The first property corresponds to the standard IS definition. The second captures spectrum reuse. In the following lemma, we show that each feasible scheduling policy maps to a RIS in G.

Lemma 5.1. *Each feasible scheduling policy maps to a RIS of G.*

Proof Sketch. Consider any feasible policy mapping to a subset of vertices in $\mathcal{V}$. We first derive a contradiction if the subset of vertices contains two neighboring vertices, indicating that each feasible policy maps to an IS. Based on the definitions of the feasible scheduling policy and the adapted FIFO model, we then prove that the parent of any induced vertex in the subset is also in the subset. □

Therefore, if we can find a MWRIS and turn it into an optimal feasible scheduling policy, the obtained scheduling policy is an optimal solution of **P4**. We thus concentrate on the MWRIS problem formulated as below.

$$\begin{aligned} \text{maximize} \quad & \sum_{v \in \mathcal{V}} w_v y_v \\ \text{subject to} \quad & y_v + y_u \leq 1, \quad \forall uv \in \mathcal{E} && (5.1) \\ & y_u \leq y_v, \quad \forall v \prec u && (5.2) \\ & y_v \in \{0,1\}, \quad \forall v \in \mathcal{V}, \end{aligned}$$

where y_v is the binary variable indicating whether vertex v is selected. Constraint (5.1) states that any pair of neighboring vertices cannot be chosen together. Constraint (5.2) states that an induced vertex can be chosen if and only if its parent is chosen.

5.3.4 LP Relaxation

To solve the MWRIS problem formulated above, we first solve its linear programming (LP) relaxation and then round the LP solution to a RIS. In this subsection we develop an LP relaxation approach tailored to our problem. The next subsection is devoted to the rounding process. Table 5.3 lists the additional notations used in Sections 5.3.4, 5.3.5, and 5.3.6.

Table 5.3: Additional notations in Sections 5.3.4, 5.3.5, and 5.3.6

$\mathcal{V}^{\alpha}(t,s)$	set of active elementary vertices in $\mathcal{V}$ at (t,s)
$\mathcal{B}^{\alpha}(t,s)$	set of active groups at (t,s)
$\mathcal{V}^{c}(i)$	set of elementary vertices valid at slot $\max\{0, a_i + l_b - L_b\}$ corresponding to the request j with $j \leq i$ for a request i
$\mathcal{B}^{c}(i)$	set of groups, each of which has at least one vertex in $\mathcal{V}^{c}(i)$
$\mathcal{N}^{c}(i)$	set of requests, each of which has at least one vertex in $\mathcal{V}^{c}(i)$
y_v^*	value of y_v in the LP relaxation of the RIS problem
λ	$\max_{k\in\mathcal{N}}(d_k - a_k + 1)$
l_{min}	$\min_{k\in\mathcal{N}} l_k$
l_{max}	$\max_{k\in\mathcal{N}} l_k$
D_1	$(l_{max} - l_{min} + 1)^2(f_{max} - f_{min} + 1) \cdot \lfloor \frac{F}{f_{min}} \rfloor \cdot \lfloor \frac{\lambda - l_{min} + l_{max}}{l_{min}} \rfloor$
D_2	$\max_{1\leq b\leq B}(L_b + l_b)(L_b - l_b + 1)(F_b - f_b + 2)/2$
$\hat{N}$	$(F - f_{min} + 1)\delta^{1+\epsilon}$
$\mathcal{V}'$	set of auxiliary vertices generated from the vertices in $\mathcal{V}$
$\mathcal{C}$	ordered set of $(D_1 + f_{max} \cdot D_2)\hat{N} - f_{max}$ colors
Γ_v	set of vertices including v to be colored in the iteration of coloring v
Γ^*	set of vertices from which the vertices in Γ are generated
$\mathcal{V}'_{k,v}$	set of vertices generated from the vertices $u \in \mathcal{V}_k$ with $v \prec u$, for each elementary $v \in \mathcal{V}$, each request $k \in \mathcal{N}_v^+$

In the LP relaxation, for each vertex $v \in \mathcal{V}$, we replace the constraint $y_v \in \{0,1\}$ by $y_v \geq 0$. There is no need to explicitly add the constraint $y_v \leq 1$ since it is implied by the other constraints. We add the constraint $\sum_{v\in\mathcal{V}_i} y_v \leq 1$ to restrict that each request i is served at most once. For each elementary vertex v, we replace constraint (5.2) by

constraint (5.3). We can check that (5.3) holds at any feasible scheduling policy.

$$\sum_{u \in \mathcal{V}_j, v \prec u} y_u \leq y_v, \ \forall v \in \mathcal{V}_i^+, \ \forall i, j \in \mathcal{N}. \tag{5.3}$$

It is well-known that the LP relaxation of the MWIS problem suffers the so-called *half integer* effect due to the edge constraint [39]. To mitigate this effect, we add the specific constraints of our scheduling problem to the LP relaxation. In this regard, the pivotal technical challenge is to construct appropriate constraints such that (1) any feasible scheduling policy is still feasible in the relaxed problem, (2) non-feasible solutions are eliminated as many as possible to facilitate the rounding process and to ensure the quality of the rounded integer solution. Given this challenge, we use the following two types of constraints to replace the edge constraint (5.1), where each type concerns a type of edges constructed in G, and thus corresponds to the specific constraint of our problem.

Type-1 constraints. The first type of constraints, as shown in (5.4), captures the interference among requests. For each resource block $(t, s) \in [0, T] \times [0, F]$, we call a vertex v active at (t, s) if $t \in [t_v, t_v + l_v - 1]$ and $s \in [s_v, s_v + h_v - 1]$, i.e., $(t, s) \in [t_v, t_v + l_v - 1] \times [s_v, s_v + h_v - 1]$. Basically, if the corresponding request is instantiated by an active vertex v, then the resource (t, s) is still occupied by the request. Similar, we call a request active at (t, s) if it has at least one active vertex at (t, s), and a group active at (t, s) if it has at least one active request at (t, s). Let $\mathcal{V}^\alpha(t, s)$ denote the set of active elementary vertices in $\mathcal{V}$ at (t, s). Let $\mathcal{B}^\alpha(t, s)$ denote the set of active groups at (t, s). The type-1 constraints state that the size of $\mathcal{V}^\alpha(t, s)$ at any feasible scheduling policy is upper-bounded by $\max_{b \in \mathcal{B}^\alpha(t,s)} (L_b + l_b)(L_b - l_b + 1)(F_b - f_b + 1)/2$.

Type-1 constraints:

$$\sum_{v \in \mathcal{V}^\alpha(t,s)} y_v \leq \max_{b \in \mathcal{B}^\alpha(t,s)} \frac{(L_b + l_b)(L_b - l_b + 1)(F_b - f_b + 1)}{2},$$
$$\forall 0 \leq t \leq T, \ \forall 0 \leq s \leq F. \tag{5.4}$$

Lemma 5.2 proves that any feasible scheduling policy satisfies type-1 constraints. Hence, imposing them does not lose any feasible scheduling policy.

Lemma 5.2. *Any feasible scheduling policy satisfies type-1 constraints.*

Proof Sketch. Consider each group b and each (t, s) pair. We first prove that the number

of elementary vertices in $\mathcal{V}^\alpha(t,s)$ of group b at any feasible scheduling policy is upper-bounded by $(L_b+l_b)(L_b-l_b+1)(F_b-f_b+1)/2$. We then prove that the vertices in $\mathcal{V}^\alpha(t,s)$ at any feasible scheduling policy belong to a same group. Therefore, for each (t,s) pair, the size of $\mathcal{V}^\alpha(t,s)$ at any feasible scheduling policy is upper-bounded by $\max_{b\in\mathcal{B}^\alpha(t,s)}(L_b+l_b)(L_b-l_b+1)(F_b-f_b+1)/2$. □

Let $l_{min} \triangleq \min_{k\in\mathcal{N}} l_k$ and $l_{max} \triangleq \max_{k\in\mathcal{N}} l_k$. We next derive the number of type-1 constraints. For each (t',s') pair, where $t' \in [0, l_{min}-2] \cup [T-l_{min}+1, T]$ or $s' \in [0, f_{min}-2] \cup [F-f_{min}+1, F]$, without loss of generality, assume $t' \in [0, l_{min}-2]$ and $s' \in [0, F]$, the vertices active at (t',s') are also active at $(l_{min}-1, s')$. Therefore, we just need to list type-1 constraints for the pairs $(t,s) \in [l_{min}-1, T-l_{min}+1] \times [f_{min}-1, F-f_{min}+1]$. As each pair corresponds to at most one type-1 constraint, there are in total $O((F-2f_{min})(T-2l_{min}))$, asymptotically $O(FT)$, type-1 constraints.

Type-2 constraints. The second type of constraints, as shown in (5.5), captures the adapted FIFO model. For each time slot t, we call a vertex v *valid* at slot t if $t_v \geq t$. For any request i of group b, let $\mathcal{V}^c(i)$ denote the set of elementary vertices that are valid at slot $\max\{0, a_i+l_{min}-l_{max}\}$ and that correspond to requests $j \in \mathcal{N}$ with $j \leq i$, i.e., request i and requests in $\mathcal{N}$ with higher priority than i. For each elementary vertex u of request k, based on the construction of G, the vertices, which are connected with u by type 2 edges, are valid at slot $\max\{0, a_k+l_k-l_v\}$. It follows from $l_k \geq l_{min}, l_v \leq l_{max}$ that $a_k+l_k-l_v \geq a_k+l_{min}-l_{max}$. Therefore, $\mathcal{V}^c(i)$ includes all elementary vertices v satisfying: (1) v is connected with one of the vertices in $\mathcal{V}_i$ by a type 2 edge, and (2) $\beta_v \leq i$. Let $\mathcal{B}^c(i)$ denote the set of groups, each of which has at least one vertex in $\mathcal{V}^c(i)$. Let $\mathcal{N}^c(i)$ denote the set of requests, each of which has at least one vertex in $\mathcal{V}^c(i)$.

For each group $b \in \mathcal{B}^c(i)$, Lemma 5.3 proves that the number of vertices in $\mathcal{V}^c(i)$ belonging to group b is upper-bounded by $(l_{max}-l_{min}+1)^2(f_{max}-f_{min}+1)$ at any feasible scheduling policy.

Lemma 5.3. *For each group $b \in \mathcal{B}^c(i)$, there are at most $(l_{max}-l_{min}+1)^2(f_{max}-f_{min}+1)$ vertices in $\mathcal{V}^c(i)$ belonging to group b at any feasible scheduling policy.*

Proof Sketch. Consider each group $b \in \mathcal{B}^c(i)$. For each slot $\tau \in [\max\{0, a_i+l_{min}-l_{max}\}, a_i)$, we first prove that the number of vertices $v \in \mathcal{V}^c(i)$ belonging to group b with $t_v = \tau$ is upper-bounded by $(l_{max}-l_{min}+1)(f_{max}-f_{min}+1)$ at any feasible scheduling policy. Then, we prove that the number of vertices $v \in \mathcal{V}^c(i)$ belonging to

group b with $t_v \geq a_i$ is upper-bounded by $(l_{max} - l_{min} + 1)(f_{max} - f_{min} + 1)$ at any feasible scheduling policy. By combining the above analysis, the number of vertices in $\mathcal{V}^c(i)$ belonging to group b is upper-bounded as $(l_{max} - l_{min} + 1)^2(f_{max} - f_{min} + 1)$ at any feasible scheduling policy. □

Let $\lambda \triangleq \max_{k\in\mathcal{N}}(d_k - a_k + 1)$. Because the spectrum allocated to any pair of requests belonging to different groups cannot overlap, the requests in $\mathcal{N}^c(i)$ that are served by allocating the time-frequency blocks in the spectrum $[\max\{0, a_i + l_{min} - l_{max}\}, \max_{k\in\mathcal{N}^c(i)} d_k] \times [0, F]$ are from at most $\lfloor \frac{F}{f_{min}} \rfloor \cdot \lfloor \frac{\max_{k\in\mathcal{N}^c(i)} d_k - a_i - l_{min} + l_{max} + 1}{l_{min}} \rfloor \leq \lfloor \frac{F}{f_{min}} \rfloor \cdot \lfloor \frac{\lambda - l_{min} + l_{max}}{l_{min}} \rfloor$ groups at any feasible scheduling policy, indicating the following constraints.

$$\text{Type-2 constraints: } \sum_{v\in\mathcal{V}^c(i)} y_v \leq D_1, \quad \forall i \in \mathcal{N}, \tag{5.5}$$

where $D_1 = (l_{max} - l_{min} + 1)^2(f_{max} - f_{min} + 1) \cdot \lfloor \frac{F}{f_{min}} \rfloor \cdot \lfloor \frac{\lambda - l_{min} + l_{max}}{l_{min}} \rfloor$.

We now derive the complexity to establish type-2 constraints. For each user i, there is at most one type-2 constraint. The complexity for listing all vertices in $\mathcal{V}^c(i)$ is upper-bounded by $\sum_{k\in\mathcal{N}, k\leq i} \delta_k(F - f_k + 1)$. Hence, listing type-2 constraints takes $O(N \sum_{k\in\mathcal{N}, k\leq i} \delta_k(F - f_k + 1)) = O(N \sum_{i\in\mathcal{N}} \delta_k(F - f_k + 1))$, asymptotically $O(\delta NF)$, time.

By combining the above analysis, we can replace the constraints (5.1) and (5.2) by constraints (5.3), (5.4), (5.5) to transform the LP relaxation of the RIS problem to the LP problem below, denoted by **P4'**.

$$\begin{aligned}
\textbf{P4':} \quad \text{maximize} \quad & \sum_{v\in\mathcal{V}} w_v y_v \\
\text{subject to} \quad & (5.3), (5.4), (5.5) \\
& \sum_{v\in\mathcal{V}_i} y_v \leq 1, \quad \forall i \in \mathcal{N} \\
& y_v \geq 0, \quad \forall v \in \mathcal{V}
\end{aligned}$$

We do not have to explicitly add the constraint $y_v \leq 1$ since it is implied by the other constraints. We have shown that any feasible scheduling policy is a feasible solution of **P4'**. Hence, the value of an optimal fractional solution of **P4'** is an upper bound of the feasible scheduling policy.

5.3.5 Approximation Scheduling Algorithm

Our approximation scheduling algorithm first solves **P4'**, then applies the rounding and coloring technique developed in [7] adapted to our context. To make our presentation streamlined and self-contained, we present and analyze the adapted rounding algorithm in the context of our problem. At a high level, our algorithm constructs a set of auxiliary vertices based on the rounded solution of **P4'** and then colors the auxiliary vertices such that each color induces a RIS. We further adjust the optimal RIS among the constructed RISes such that the set of resulting vertices still forms a RIS and thus maps to a feasible scheduling policy. The algorithm is composed of three steps, which are exposed sequentially. The first two steps essentially follow the procedures in [7], which are adapted below in the context of our problem. In the third step, we adjust the RIS such that the vertex set still forms a RIS and maps to a feasible scheduling policy.

Some additional notations are used in this subsection. Let $D_2 \triangleq \max_{1\leq b\leq B}(L_b + l_b)(L_b - l_b + 1)(F_b - f_b + 2)/2$. Let $\hat{N} \triangleq (F - f_{min} + 1)\delta^{1+\epsilon}$. To make our analysis concise, we assume that $\hat{N}$ is an integer, otherwise we need to round it to the nearest integer. Let $f_{max} \triangleq \max_{k\in\mathcal{N}} f_k$, and let $\mathcal{C}$ denote an ordered set (or a vector) of $(D_1 + f_{max} \cdot D_2)\hat{N} - f_{max}$ colors.

Step 1: Generating the auxiliary vertices. We solve **P4'** and denote the solution by $\{y_v^*\}_{v\in\mathcal{V}}$. Clearly, we have $0 \leq y_v^* \leq 1, \forall v \in \mathcal{V}$. For each vertex $v \in \mathcal{V}$, we create a set of $\lfloor y_v^*\hat{N} \rfloor$ new vertices, termed as *auxiliary vertices*. Each generated vertex is a duplicate of v, corresponding to the same rectangle, having the same priority and covering the same request as v. Let $\mathcal{V}'$ denote the set of auxiliary vertices generated from $\mathcal{V}$. For each elementary $v \in \mathcal{V}$ and each request $k \in \mathcal{N}_v^+$, let $\mathcal{V}'_{k,v}$ denote the set of auxiliary vertices in $\mathcal{V}'$ generated from the induced vertices $u \in \mathcal{V}_k$ with $v \prec u$. We say that the vertices in $\mathcal{V}'_{k,v}$ are the *auxiliary children* of v covering request k.

Step 2: Coloring the auxiliary vertices. We color each vertex in $\mathcal{V}'$ by a color such that (1) any pair of vertices generated from a same vertex in $\mathcal{V}$ is not of the same color; (2) any pair of vertices that are generated from the adjacent vertices in $\mathcal{V}$ is not of the same color; and (3) an induced vertex has the same color as an elementary vertex that is generated from its parent. Specifically, we first sort the elementary vertices in $\mathcal{V}'$ such that for any pair of elementary vertices $u, v \in \mathcal{V}'$, v is left to u if $t_v < t_u$ or $t_v = t_u, s_v < s_u$, with ties broken randomly. Step 2 is then executed in iterations. In each iteration, we try to color an elementary vertex in $\mathcal{V}'$. For each elementary vertex $u \in \mathcal{V}'$ from left to right, we stock the set of vertices to be colored in the current

iteration in Γ, which is initialized to $\{u\}$. Let v denote the vertex in $\mathcal{V}$ from which u is generated. If $\mathcal{N}_v^+ \neq \emptyset$, for each request $k \in \mathcal{N}_v^+$, if there is an uncolored induced vertex in $\mathcal{V}'_{k,v}$, we add the vertex to Γ. We finalize the coloring process of the elementary vertex u by coloring all vertices in Γ using the first color in $\mathcal{C}$ that has not been used to color any vertex generated from the vertices in Γ^* and the neighbors of Γ^*, where Γ^* stocks the set of vertices in $\mathcal{V}$ from which the vertices in Γ are generated.

Step 3: Adjusting the RIS. Based on the coloring result obtained in Step 2, we choose a set of vertices of the same color with maximum total weight, and let $\hat{\mathcal{V}}$ denote the set of vertices in $\mathcal{V}$ from which the chosen vertices are generated. We prove that the set of vertices, from which the vertices of the same color are generated, is a RIS in Lemma 5.5. However, $\hat{\mathcal{V}}$ may not map to a feasible policy. In this case, we adjust $\hat{\mathcal{V}}$ such that it forms a RIS and maps to a feasible scheduling policy. Consider a RIS mapping to a feasible scheduling policy. For each vertex u in the RIS, let i denote the request covered by u. Based on the definition of the adapted FIFO model, u uses or reuses the time-frequency block of the vertex with highest priority among u and the elementary vertices v in the RIS with $i \in \mathcal{N}_v^+$. Technically, for each vertex $\nu \in \hat{\mathcal{V}}$, let k denote the request covered by ν. If ν is an elementary vertex, we prove that there is no elementary vertex $v \in \hat{\mathcal{V}}$ with $\beta_v < \beta_\nu$ and $k \in \mathcal{N}_v^+$ in Lemma 5.6. If ν is an induced vertex, we first seek the set of elementary vertices $v \in \hat{\mathcal{V}}$ with $k \in \mathcal{N}_v^+$, and denote the set of vertices by Φ. Since $\hat{\mathcal{V}}$ is a RIS, the parent of ν also belongs to Φ. We then pick the vertex with highest priority among the vertices in Φ, denoted by μ. Because the parent of ν belongs to Φ and μ has the highest priority among the vertices in Φ, the priority of μ is higher than or equal to that of ν, i.e., $\beta_\mu \leq \beta_\nu$. If $\beta_\mu = \beta_\nu$, request k is instantiated by ν by reusing the time-frequency block of μ; if $\beta_\mu < \beta_\nu$, request k should be served by reusing the time-frequency block of μ under the priority of μ, and we thus replace ν by an induced vertex in $\mathcal{V}_k$ whose parent is μ in $\hat{\mathcal{V}}$. It follows from the construction of vertices in G that there is at least one induced vertex in $\mathcal{V}_k$ whose parent is μ as $k \in \mathcal{N}_\mu^+$.

Example 5.4. *We illustrate Step 3 by reconsidering Example 5.2, where we select* $\hat{\mathcal{V}} = \{v_1, v_3, v_8, v_{12}, v_{13}\}$. *It can be noted that* $\hat{\mathcal{V}}$ *is a RIS of* G *but does not map to a feasible scheduling policy, we replace* v_8 *by* v_4 *in* $\hat{\mathcal{V}}$. *Now* $\hat{\mathcal{V}}$ *maps to the unique optimal scheduling policy.*

We summarize that the key technicality in our algorithm is to color each induced vertex using the same color as its parent, and then adjust the RIS to obtain a feasible

scheduling policy.

The pseudo-code of our algorithm is given in Algorithm 5. As each vertex corresponds to a request, the final scheduling policy is to serve the requests corresponding to the set of vertices output by Algorithm 5. In the following, we briefly describe the functions used in our algorithm, most of which are graph algorithms that can be coded straightforwardly. The detailed implementation is thus omitted in the pseudo-code.

- **AuxiliaryChildren**$(v, k, \mathcal{V}')$ returns the set of auxiliary vertices in $\mathcal{V}'$ generated from the induced vertices in $\mathcal{V}_k$ whose parents are v, given the elementary vertex $v \in \mathcal{V}$, the request $k \in \mathcal{N}_v^+$ and the set of auxiliary vertices $\mathcal{V}'$.
- **SimultaneousColoredVertex**$(\mathcal{V}'_{k,v})$ returns an uncolored vertex in $\mathcal{V}'_{k,v}$ if such vertex exists, $\emptyset$ otherwise, given the set of vertices $\mathcal{V}'_{k,v}$.
- **GeneratedfromVertices**$(\mathcal{V}^*, \mathcal{V})$ returns the set of vertices in $\mathcal{V}$, from which the vertices in $\mathcal{V}^*$ are generated, given the sets of vertices in $\mathcal{V}$ and $\mathcal{V}^* \subseteq \mathcal{V}'$.
- **UsedColors**(Γ^*) returns the set of colors that have been used to the colored vertices generated from the vertices in Γ^* and the neighbors of Γ^*, given the sets of vertices in Γ^*.
- **Color**$(\mathcal{C}^*, \mathcal{C})$ returns the first color in $\mathcal{C} \setminus \mathcal{C}^*$, given the sets of colors in $\mathcal{C}$ and $\mathcal{C}^*$.
- **MaxWeightVertices**$(\mathcal{V}')$ returns the set of vertices in $\mathcal{V}'$ of the same color with maximum total weight, given the set of vertices in $\mathcal{V}'$.
- **FittedVertices**$(i, \hat{\mathcal{V}})$ returns the set of elementary vertices $v \in \hat{\mathcal{V}}$ with $i \in \mathcal{N}_v^+$, given the request i, and the set of vertices in $\hat{\mathcal{V}}$.
- **HighestPrioVertex**(Φ) returns the vertex with the highest priority among the vertices in Φ, given the set of vertices in Φ.
- **InducedVertex**$(\mu, \mathcal{V}_i)$ returns an induced vertex in $\mathcal{V}_i$ whose parent is μ, given the elementary vertex v and the set of vertices in $\mathcal{V}_i$.

Algorithm 5 Offline scheduling policy: executed by the scheduler

1: **Input:** request graph $G = (\mathcal{V}, \mathcal{E})$, solution of the LP relaxation $\{y_v^*\}_{v \in \mathcal{V}}$, $\hat{N} = (F - f_{min} + 1)\delta^{1+\epsilon}$, a vector $\mathcal{C}$ of $(D_1 + f_{max} \cdot D_2)\hat{N} - f_{max}$ colors
2: **Output:** set of vertices $\hat{\mathcal{V}}$ ▷ The corresponding policy is to serve the requests corresponding to $\hat{\mathcal{V}}$
3: $\mathcal{V}' \leftarrow \emptyset$ ▷ $\mathcal{V}'$ stocks the set of auxiliary vertices generated from $\mathcal{V}$

for each vertex $v \in \mathcal{V}$ **do** ▷ Construct the auxiliary vertices $\mathcal{V}'$
    construct $\lfloor y_v^* \hat{N} \rfloor$ new vertices, each of which is a duplicate of v, corresponding to the same rectangle, having the same priority and covering the same request as v
    add the constructed vertices to $\mathcal{V}'$
end for

for each elementary $v \in \mathcal{V}$ **do**
    for each request $k \in \mathcal{N}_v^+$ **do**
        $\mathcal{V}'_{k,v} \leftarrow$ **AuxiliaryChildren**$(v, k, \mathcal{V}')$
    end for
end for

sort the elementary vertices v in $\mathcal{V}'$ lexicographically based on (t_v, s_v) with ties broken randomly
for each elementary vertex $u' \in \mathcal{V}'$ from left to right **do**
    let u denote the vertex from which u' is generated
    $\Gamma \leftarrow \{u'\}$ ▷ Γ stocks the set of vertices to be colored in the current iteration
    if $\mathcal{N}_u^+ \neq \emptyset$ **then**
        for each $k \in \mathcal{N}_u^+$ **do**
            $r \leftarrow$ **SimultaneousColoredVertex**$(\mathcal{V}'_{k,u})$
            add vertex r to Γ
        end for
    end if
    $\Gamma^* \leftarrow$ **GeneratedfromVertices**$(\Gamma, \mathcal{V})$
    $\mathcal{C}^* \leftarrow$ **UsedColors**(Γ^*)
    color all vertices in Γ using **Color**$(\mathcal{C}^*, \mathcal{C})$
end for

$\hat{\mathcal{V}} \leftarrow$ **GeneratedfromVertices**(**MaxWeightVertices**$(\mathcal{V}')$,$\mathcal{V}$)
for each induced vertex $\nu \in \hat{\mathcal{V}}$ **do**
    let i denote the request corresponding to ν
    $\Phi \leftarrow$ **FittedVertices**$(i, \hat{\mathcal{V}})$ ▷ Φ stocks the set of elementary vertices $u \in \hat{\mathcal{V}}$ with $i \in \mathcal{N}_u^+$
    $\mu \leftarrow$ **HighestPrioVertex**(Φ)
    if $\beta_\mu < \beta_\nu$ **then** ▷ i should be served by reusing μ's time-frequency block under the priority of μ
        replace ν by **InducedVertex**$(\mu, \mathcal{V}_i)$ in $\hat{\mathcal{V}}$
    end if
end for
return set of vertices in $\hat{\mathcal{V}}$

5.3.6 Performance Analysis

In this subsection we derive the theoretical performance guarantee of our approximation algorithm. We first prove that all vertices in $\mathcal{V}'$ are colored by Algorithm 5 in Lemma 5.4. Then, we prove that the set of vertices, from which the vertices of any

same color are generated, induces a RIS in Lemma 5.5, and Algorithm 5 outputs a set of vertices mapping to a feasible scheduling policy in Lemma 5.6. We are then able to establish the approximation factor of Algorithm 5 in Theorem 5.2. We conclude this subsection by giving the complexity of Algorithm 5.

Lemma 5.4. *All vertices in $\mathcal{V}'$ are colored by Algorithm 5.*

Proof Sketch. We prove that any elementary vertex and any induced vertex is colored by Algorithm 5. □

Lemma 5.5. *For any color $c \in \mathcal{C}$, the set of vertices, from which the vertices colored by c are generated, induces a RIS.*

Proof Sketch. We first prove that any pair of vertices colored by c is not generated from a same vertex or any pair of adjacent vertices, indicating that the set of vertices, from which the vertices colored by c are generated, is not adjacent to each other, i.e., the set of vertices satisfies the first property of RIS. For each induced vertex colored by c, let u denote the vertex from which the vertex is generated. We then prove that there is a vertex, which is generated from the parent of u, colored by c, indicating that the set of vertices, from which the vertices colored by c are generated, satisfies the second property of RIS. By combining the above analysis, the set of vertices, from which the vertices colored by c are generated, induces a RIS. □

Lemma 5.6. *Algorithm 5 outputs a set of vertices that maps to a feasible scheduling policy.*

Proof Sketch. By Lemma 5.5, the set of vertices, denoted by $\hat{\mathcal{V}}$, output by Algorithm 5 is a RIS of G. We prove that the set of vertices in $\hat{\mathcal{V}}$ follows the adapted FIFO model. It then follows from the definition of feasible scheduling policy that the output of Algorithm 5 maps to a feasible scheduling policy. □

Theorem 5.2. *Algorithm 5 outputs an asymptotically $1/(D_1+f_{max}\cdot D_2)$-optimal feasible scheduling policy for the offline scheduling problem, i.e., Algorithm 5 is a $1/(D_1+f_{max}\cdot D_2)$-approximation algorithm.*

Proof Sketch. By Lemma 5.4 and Lemma 5.5, we get at most $(D_1 + f_{max} \cdot D_2)\hat{N} - f_{max}$ RISes of G. We prove that there exists a RIS whose total reward is at least $1/(D_1 + f_{max} \cdot D_2)$ of the utility for the optimal feasible scheduling policy. The theorem is thus proved. □

Theorem 5.2 demonstrates that the approximation factor of our algorithm depends on the largest request slackness and parameters $l_b, L_b, f_b, F_b, l_{max}, l_{min}, f_{max}, f_{min}, F$, where F is a constant. Thus, the performance does not degrade with the system size.

We conclude the analysis by giving the complexity of our approximation algorithm. We first need to compute the number of vertices in $\mathcal{V}'$. To that end, we can calculate the number of created vertices in $\mathcal{V}$ that is $O(RMF\delta)$. The total number of vertices in $\mathcal{V}'$ sums to $|\mathcal{V}'| = O(RMF\delta\hat{N})$. The complexity for the sorting of the elementary vertices in $\mathcal{V}'$ is $O(F\delta\hat{N}\log(F\delta\hat{N}))$ since there are at most $F\delta\hat{N}$ elementary vertices in $\mathcal{V}'$. Therefore, it follows from $\hat{N} = (F - f_{min} + 1)\delta^{1+\epsilon}$ that the complexity of Algorithm 5 is $O(RMF^2\delta^{2+\epsilon} + F^2\delta^{2+\epsilon}\log(F\delta))$.

5.4 The Online Case

In this section, we consider the online scheduling problem, where the scheduler only knows the current scheduling backlog. More specifically, a_i, d_i, f_i, l_i, b_i, and w_i are known only at the moment when request i arrives. As the offline case, the online scheduling problem also takes into account the adapted FIFO service model and the spectrum reusability. Table 5.4 lists the additional notations in the online setting.

Table 5.4: Additional notations in Section 5.4

s_i	frequency to start serving i
t_i	starting time to serve request i
Ω_t	set of active requests at the current slot
$\mathcal{B}(\Omega_t)$	set of groups containing at least a request in Ω_t
B_{max}	$\max_{0\leq t\leq T} \lvert\mathcal{B}(\Omega_t)\rvert$
$\mathcal{N}_t^*$	set of requests started being served at slot t by Algorithm 6
$\mathcal{N}_t^-$	set of requests started being served earlier than slot t
$\mathcal{T}$	set of slots, at which Algorithm 6 starts serving at least one request under the priority of itself

In the online setting, we focus on the *non-preemptive* scheduling model: once a request starts being served, it must be completed without interruption [19] [21]. We note that the non-preemptive model is seamlessly compatible with our adapted FIFO model. Our analysis can be extended to the other models such as the preemptive-resume and the preemptive-restart models.

5.4.1 Inapproximability

We start by showing that the online problem in its generic form cannot be approximated with any finite competitive ratio.

Theorem 5.3. *For any $\rho > 0$, there exists an instance of our scheduling problem, where the competitive ratio of any deterministic online algorithm Π on the instance is larger than ρ.*

Proof. We consider the following instance of our online scheduling problem. In slot 0, request 1 belonging to group 1 with the quadruple $(0, 1, 1, 2)$ arrives, whose reward is 1. We distinguish two cases.

- If Π does not serve the request, then there is no more request arriving, leading to a competitive ratio of infinity as the optimal solution is clearly to serve the request;
- If Π serves the request, a set of requests belonging to group 2 arrives, each characterized by the same quadruple $(1, 2l_{max}/l_{min}, F, 2l_{max}/l_{min})$, and the total reward of them is more than ρ. Under this request sequence, the optimum scheduling policy serves the requests belonging to group 2 while Π only serves the request belonging to group 1, leading to a competitive ratio larger than ρ.

Combining the above two cases completes our proof. □

5.4.2 Online Scheduling Algorithm Design

Theorem 5.3 establishes the inapproximability of the online case of our scheduling problem in the generic case. By examining the problem instance in Theorem 5.3, we observe that the inapproximability is due to the stringent deadline constraint in the instance. Consequently, any online algorithm is forced to make a decision immediately when the requests arrive. Motivated by this observation, we slightly relax the slackness of the requests, and design an online scheduling algorithm with bounded competitive ratio. Specifically, we assume that the slackness of each request i is at least $l_{max} + l_i - 1$, i.e., $d_i - a_i + 1 \geq l_{max} + l_i - 1, \forall i \in \mathcal{N}$.

We give a high-level overview of our online scheduling algorithm. At each slot t, our algorithm starting serving the requests at slot t can be divided into two cases: (1) if there is at least one request that is being served at slot t, we start serving each request i at slot t by reusing the time-frequency block of a request with highest priority among

the served requests such that i can reuse the time-frequency block of the requests if there is at least one such request, (2) if there is no request that is being served at slot t, we start serving the requests at slot t belonging to one group after another according to the total reward of the requests for each group under the constraint of the amount of available frequency band.

Technically, we introduce the following definitions. We say a slot *occupied* if there exists a request being served at the slot. For each slot t, we say a request k *ready* at slot t satisfying: (1) $a_k \leq t \leq d_k - l_k + 1$, (2) k has not been served before slot t, and (3) starting serving k at slot t does not violate the adapted FIFO model. For each slot t, let Ω_t denote the set of ready requests at slot t. Let $\mathcal{B}(\Omega_t)$ denote the set of groups, each of which has at least one request in Ω_t. Let $\mathcal{N}_t^-$ denote the set of requests that start being served earlier than slot t. We use a quadruple (i, t_i, s_i, β_i) to denote each served request i, where t_i and s_i are the starting time and the starting spectrum to serve request i, and β_i is the priority to serve i.

The pseudo-code of our online algorithm is given in Algorithm 6. Consider each slot t where there exists at least one ready request, i.e., $\Omega_t \neq \emptyset$, we proceed by distinguishing the following two cases.

- **Slot t is occupied**. For each request $j \in \Omega_t$, if there is at least one request $k \in \mathcal{N}_t^-$ of group b_j such that j fits into the time-frequency block $[t, t_k + l_k - 1] \times [s_k, s_k + f_k - 1]$, we pick the request with highest priority among the requests, denoted by i; we serve j by reusing the time-frequency block of i under the priority of i.

- **Slot t is not occupied**. Algorithm 6 runs in iterations. In each iteration, we try to start serving the requests in Ω_t belonging to a same group at slot t. Let f^* denote the amount of spectrum already occupied by the requests that start being served at the current slot. We first pick the group such that the total reward of the requests $k \in \Omega_t$ with $f_k \leq F - f^* + 1$ of group b is maximum among all the groups in $\mathcal{B}(\Omega_t)$. We denote the selected group by b. Mathematically,

$$b = \operatorname*{argmax}_{b \in \mathcal{B}(\Omega_t)} \sum_{k \in \Omega_t, b_k = b, f_k \leq F - f^* + 1} w_k.$$

  We then serve the requests $k \in \Omega_t$ of group b with $f_k \leq F - f^* + 1$ in rounds. In each round, we first pick the request i of the highest priority among the requests

$k \in \Omega_t$ with $f_k \leq F - f^* + 1$ of group b, i.e.,

$$i = \min_{k \in \Omega_t, b_k = b, f_k \leq F - f^* + 1} k.$$

We serve each request $k \in \Omega_t$ of group b with $l_k \leq l_i, f_k \leq f_i$ by allocating the time-frequency block $[t, t + l_k - 1] \times [f^*, f^* + f_k - 1]$ under the priority of i, and remove k from Ω_t. We continue starting serving requests of group b at slot t until there is no request $k \in \Omega_t$ of group b satisfying $f_k \leq F - f^* + 1$.

We briefly describe the following functions used in our algorithm, which can be coded straightforwardly. The detailed implementation is thus omitted in the pseudo-code.

- **FittedRequests**$(i, \mathcal{N}_t^-)$ returns the set of requests $k \in \mathcal{N}_t^-$ of group b_i such that i fits into the time-frequency block $[t, t_k + l_k - 1] \times [s_k, s_k + f_k - 1]$ if such requests exist, and $\emptyset$ otherwise, given the request i and the set of requests $\mathcal{N}_t^-$.

- **HighestPrioRequest**(Φ) returns the highest priority request in Φ, given the set of requests in Φ.

- **MaxRewardGroup**(Ω_t, f^*) returns the group such that the set of requests $k \in \Omega_t$ of the group with $f_k \leq F - f^* + 1$ has the maximum total reward among the groups in $\mathcal{B}(\Omega_t)$, given the set of requests Ω_t, and the amount of spectrum f^* already occupied by the requests that start being served at the current slot.

Algorithm 6 Online scheduling policy: executed by the scheduler at each slot t

Input: Ω_t: the set of ready requests at slot t; $\mathcal{N}_t^-$: the set of requests already starting being served
Output: $\mathcal{N}_t^*$ ▷ $\mathcal{N}_t^*$ denotes the set of requests that start being served at slot t, and each request $i \in \mathcal{N}_t^*$ is served by allocating the resource block $[t, t + l_i - 1] \times [s_i, s_i + f_i - 1]$ under the priority of β_i
Initialization: $f^* = 0$, $\mathcal{N}_t^* \leftarrow \emptyset$
if $\Omega_t == \emptyset$ **then**
    return $\emptyset$
end if
if slot t is occupied **then**
    for each request $k \in \Omega_t$ **do**
        $\Psi \leftarrow$ **FittedRequests**$(k, \mathcal{N}_t^-)$
        if $\Psi \neq \emptyset$ **then**
            $i \leftarrow$ **HighestPrioRequest**(Ψ)
            $t_k = t, s_k = f_i, \beta_k = i$ ▷ Request k is served by reusing the time-frequency block of i

add (k, t_k, s_k, β_k) to $\mathcal{N}_t^*$
end if
end for
return $\mathcal{N}_t^*$
end if
while $\Omega_t \neq \emptyset$ **do**
$b \leftarrow$ **MaxRewardGroup**(Ω_t, f^*)
$\Phi \leftarrow \emptyset$ ▷ Φ stocks the set of requests $k \in \Gamma$ with $b_k = b$ and $f_k \leq F - f^* + 1$
for each request $j \in \Omega_t$ **do**
if $b_j == b$ **and** $f_j \leq F - f^* + 1$ **then**
add j to Φ
end if
end for
if $\Phi == \emptyset$ **then** ▷ There is not enough spectrum to serve any request
break
end if
while $\Phi \neq \emptyset$ **do**
$i \leftarrow$ **HighestPrioRequest**(Φ)
for each request $k \in \Phi$ **do**
if $l_k \leq l_i, f_k \leq f_i$ **and** $b_k == b_i$ **then**
$t_k = t, s_k = f^*, \beta_k = i$ ▷ Serve request k by allocating resource block $[t, t + l_k - 1] \times [f^*, f^* + f_k - 1]$ under priority of i
add (k, t_k, s_k, β_k) to $\mathcal{N}_t^*$
remove k from Φ and Ω_t
end if
end for
end while
$f^* \leftarrow f^* + \max_{k \in \Phi} f_k$ ▷ the amount of spectrum occupied by the requests in Φ is $\max_{k \in \Phi} f_k$
end while
return $\mathcal{N}_t^*$

5.4.3 Performance Analysis

Theorem 5.4. *If the slackness of each request i is at least $l_{max} + l_i - 1$, i.e., $d_i - a_i + 1 \geq l_{max} + l_i - 1, \forall i \in \mathcal{N}$, the competitive ratio of Algorithm 6 is upper-bounded by B_{max}, i.e., Algorithm 6 is a $1/B_{max}$-competitive, where $B_{max} \triangleq \max_{0 \leq t \leq T} |\mathcal{B}(\Omega_t)|$.*

Proof Sketch. Let $\mathcal{T}$ denote the set of slots, where each slot $t \in \mathcal{T}$ satisfies that: (1) Algorithm 6 starts serving at least one request k at slot t, and (2) t is not occupied when Algorithm 6 considers the slot t. We first prove that the utility of the optimal scheduling policy is upper-bounded by the total reward of the requests, each of which

is ready at a slot in $\mathcal{T}$. For each slot $t \in \mathcal{T}$, we then derive the relationship between the total reward of the requests starting being served by Algorithm 6 at slot t and the total reward of the requests ready at slot t. Combining the above results allows us to establish the competitive ratio upper-bound of Algorithm 6. □

Theorem 5.4 shows that the performance of our algorithm only scales with the largest size of $\mathcal{B}(\Omega_t)$, which only depends on the number of packets that are requested by the ready users at the current slot t. Therefore, the global efficiency of our algorithm does not degrade with the system size.

5.5 Numerical Analysis

In this section, we conduct numerical analysis to evaluate the performance of the offline and online scheduling algorithms we develop. In our simulation, we trace the metric (3.8) to evaluate the performance of the optimal scheduling policy compared to our algorithms. Specifically, we trace the maximal, average, and minimal values of Υ in our simulations.

The time horizon T is set to 200, the amount of frequency granularity F is set to 10; $l_{max} = 10, l_{min} = 1$. We simulate three typical scenarios, in each of which we vary the number of requests N in the system from 50 to 500. For each N, we perform 50 simulation runs for each request parameter setting.

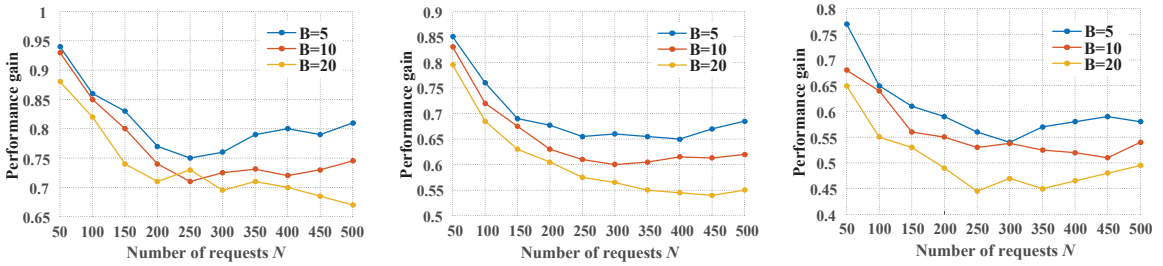
5.5.1 Scenario 1

In the first scenario, we randomly choose the parameters a_i, d_i, f_i, l_i, b_i such that $1 \leq f_i \leq F, d_i - a_i + 1 \geq l_i, \forall i \in \mathcal{N}$. We run three experiments with $B = 5, 10$ and 20 respectively. The simulation results of the offline and online cases are illustrated in Figures 5.5 and 5.6.

From the simulation results, we make the following observations.

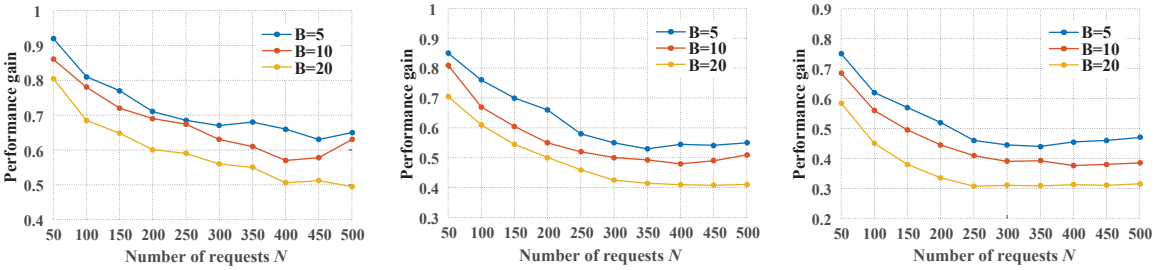
- Our algorithms achieve at least 44% of the optimal utility even in the worst case in the offline setting and at least 30% in the online setting, which are in accordance to the theoretical results we derive.

- Our offline algorithm performs better compared to our online algorithm, as the spectrum broker disposes more information in the offline setting and naturally achieves better performance.



(a) Maximum performance gains (b) Average performance gains (c) Minimum performance gains

Figure 5.5: Performance gains of Algorithm 5 for Scenario 1 in offline case



(a) Maximum performance gains (b) Average performance gains (c) Minimum performance gains

Figure 5.6: Performance gains of Algorithm 6 for Scenario 1 in online case

- When the number of requests N increases, the average and minimum performance gains first decrease in both offline and online cases, and then stabilize, indicating that the resource pool approaches its capacity limit.

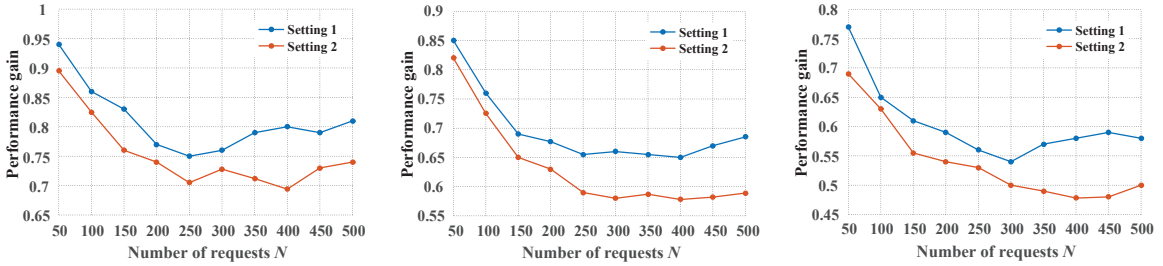
- The smaller the number of groups there is, the better performance our algorithms achieve. This is because, when the number of groups is small, there are more opportunities for spectrum reuse, which potentially improves the system performance.

5.5.2 Scenario 2

In this scenario, we set $B = 5$ and randomly choose the parameters a_i, d_i, f_i, l_i, b_i such that $d_i - a_i + 1 \leq l_i, \forall i \in \mathcal{N}$. The experiments of Scenario 2 include the following two settings: (1) $1 \leq f_i \leq F, \forall i \in \mathcal{N}$, and (2) $f_i = F, \forall i \in \mathcal{N}$, i.e., each user requests the whole frequency band. The simulation results of this scenario are illustrated in Figures 5.7 and 5.8. From the results, we make the following observations.

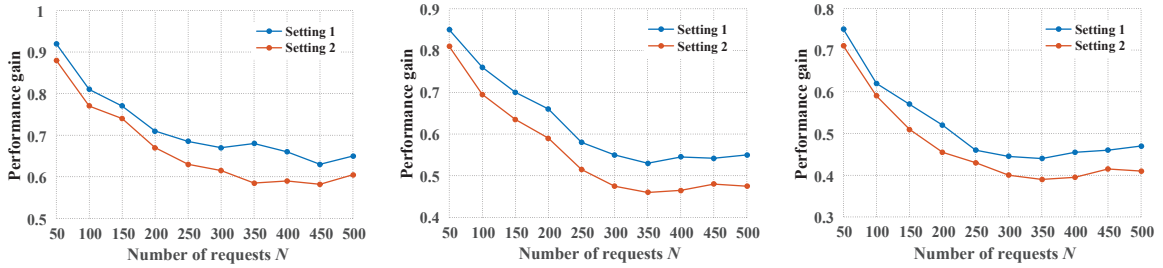
- Our algorithms achieve at least 47% of the optimal utility in offline case, and 38% of the optimal utility in online case.

- Our algorithms achieve better performance in the first setting. This is because the quantity of contiguous frequency band requested by each user i is $f_i \leq F$ in the first setting while each user requests the whole band in the second setting, indicating that the spectrum resource can be allocated to more users in the first setting.



(a) Maximum performance gains (b) Average performance gains (c) Minimum performance gains

Figure 5.7: Performance gains of Algorithm 5 for Scenario 2 in offline case



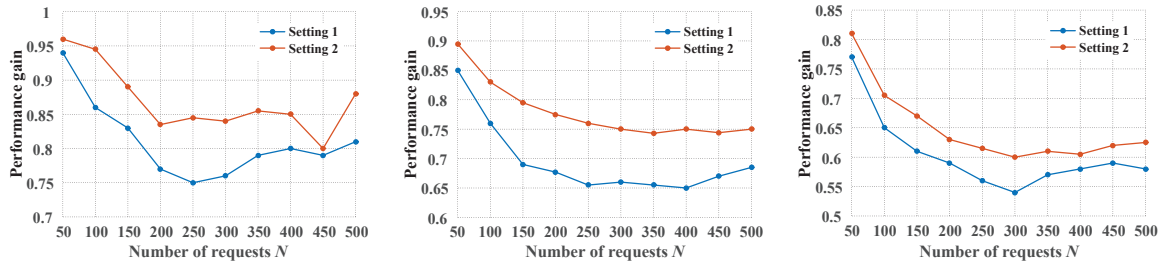
(a) Maximum performance gains (b) Average performance gains (c) Minimum performance gains

Figure 5.8: Performance gains of Algorithm 6 for Scenario 2 in online case

5.5.3 Scenario 3

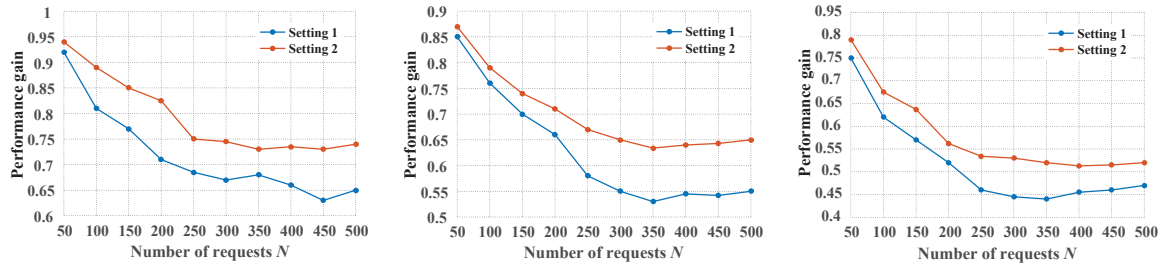
In this scenario, we set $B = 5$, and randomly choose the parameters a_i, d_i, f_i, l_i, b_i such that $1 \leq f_i \leq F, \forall i \in \mathcal{N}$. We run the experiments including the following two settings: (1) $d_i - a_i + 1 \geq l_i$, and (2) $d_i - a_i + 1 \geq l_{max} + l_i - 1$. The simulation results for this scenario are shown in Figure 5.9 and 5.10. From the results, we derive the following observations.

- Our algorithms achieve at least 54% of the optimal utility in offline case, and 43% of the optimal utility in online case.



(a) Maximum performance gains (b) Average performance gains (c) Minimum performance gains

Figure 5.9: Performance gains of Algorithm 5 for Scenario 3 in offline case



(a) Maximum performance gains (b) Average performance gains (c) Minimum performance gains

Figure 5.10: Performance gains of Algorithm 6 for Scenario 3 in online case

- By comparing the performance gains between the two settings, as the slackness of each request is relaxed, our algorithms in the second setting perform better compared to the first setting.

5.6 Conclusion and Perspective

Motivated by the spectrum bonding problem, we have formulated and analyzed the contiguous-resource batching task scheduling problem arising in a variety of engineering fields, where communication and storage resources are potential bottlenecks and thus need to be carefully scheduled. We have investigated both offline and online scheduling settings. In both cases, we have established the problem hardness and developed approximation algorithms with proven performance guarantee in terms of approximation and competitive ratios, respectively. We have complemented our theoretical analysis with numerical simulations that demonstrate the effectiveness of our algorithms in a variety of system settings.

5.7 Appendix

5.7.1 Integer Linear Problem Formulation of Offline Channel Bonding

The offline optimum spectrum bonding problem **P4** can be formulated as a network utility maximization (NUM) problem as below.

P4: maximize $\sum_{1\leq i\leq N} w_i x_i$

subject to

$$t_j - t_i \geq l_i \text{ or } s_j - s_i \geq f_i,\ \forall i,j \in \mathcal{N}, b_i \neq b_j, t_i < t_j, s_i < s_j, x_i = x_j = 1 \quad (5.6a)$$

$$t_j - t_i \geq l_i \text{ or } s_i - s_j \geq f_j,\ \forall i,j \in \mathcal{N}, b_i \neq b_j, t_i < t_j, s_i > s_j, x_i = x_j = 1 \quad (5.6b)$$

$$t_i - t_j \geq l_j \text{ or } s_j - s_i \geq f_i,\ \forall i,j \in \mathcal{N}, b_i \neq b_j, t_i > t_j, s_i < s_j, x_i = x_j = 1 \quad (5.6c)$$

$$t_i - t_j \geq l_j \text{ or } s_i - s_j \geq f_j,\ \forall i,j \in \mathcal{N}, b_i \neq b_j, t_i > t_j, s_i > s_j, x_i = x_j = 1 \quad (5.6d)$$

$$t_j \geq t_i,\ \forall i,j \in \mathcal{N}, \beta_j > \beta_i, x_i = x_j = 1 \quad (5.6e)$$

$$t_j = \max\{t_i, a_j\}, \begin{array}{r} \forall j \in \mathcal{N}, \beta_j = \beta_i, x_j = 1, i = \min\{k : t_j + l_j \leq t_k + l_k, \\ b_k = b_j, f_j \leq f_j, x_k = 1, j \geq k\} \end{array} \quad (5.6f)$$

$$0 \leq s_i \leq F - f_i,\ \forall i \in \mathcal{N}, x_i = 1 \quad (5.6g)$$

$$a_i \leq t_i \leq d_i - l_i + 1,\ \forall i \in \mathcal{N}, x_i = 1 \quad (5.6h)$$

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

where t_i is the time to start serving user i if its request is admitted, $[s_i, s_i + f_i - 1]$ is the allocated spectrum, x_i is the decision variable indicating whether to serve user i. Constraints (5.6a)-(5.6d) imply that if the user i cannot be served simultaneously with user j, their allocated spectrum cannot overlap. Constraint (5.6e) implies that if request j is served under the lower priority than the priority under which request i is served, the time to start serving j cannot be earlier that the time to start serving i. Constraints (5.6f) indicates that if request j is served, it must be served under the priority of itself or batched with a request with highest priority among the served users who belong to group b_j and whose time-frequency blocks can fit for serving j. Constraints (5.6g) and (5.6h) are the feasibility constraints in the frequency and time domains.

Table 5.5 lists the additional notations in Appendix.

Table 5.5: Additional notations in the Appendix

x_i	decision variable indicating whether to serve user i
$\mathcal{V}^{\alpha}_{b,\tau}(t,s)$	set of elementary vertices in $\mathcal{V}^{\alpha}(t,s)$ of group b whose left boundaries are slot τ
$\mathcal{N}^{\alpha}_{b,\tau}(t,s)$	set of requests having at least one vertex in $\mathcal{V}^{\alpha}_{b,\tau}(t,s)$
$\mathcal{V}^{c}_{\tau}(i)$	set of elementary vertices in $v \in \mathcal{V}^{c}(i)$ with $t_v = \tau$
$\mathcal{N}^{c}_{\tau}(i)$	set of requests having at least one vertex in $\mathcal{V}^{c}_{\tau}(i)$
$\mathcal{O}$	set of requests served by the optimal scheduling policy

5.7.2 Proof of Theorem 5.1

To prove its hardness, we consider the classical 0-1 Knapsack problem which is known to be NP-hard [26].

0-1 **Knapsack problem**. Given a positive capacity C and a set $\mathcal{N}$ of N elements indexed from 1 to N, each with a positive weight w_i and size c_i ($i \in \{1, 2, \cdots, N\}$), find a set $\hat{\mathcal{N}} \subseteq \{1, 2, \cdots, N\}$ such that $\sum_{i \in \hat{\mathcal{N}}} c_i \leq C$ and $\sum_{i \in \hat{\mathcal{N}}} w_i$ is maximized. The problem is known to be NP-hard.

We now show that the 0-1 Knapsack problem can be reduced to **P4**. To this end, consider an instance of the 0-1 Knapsack problem defined above, we construct an instance of **P4** as follows: there are N users indexed from $i = 1$ to N, each submitting a request $(0, T, F, c_i)$, i.e., each request i needs the whole spectrum, the required service time is c_i; no spectrum reuse is possible; as all requests are submitted at time 0, the service order is determined by user indexes; the reward of each request i is w_i. It is straightforward to see that a solution of the constructed instance of **P4** can be cast to a solution of the 0-1 Knapsack problem.

It then follows from the NP-hardness of the 0-1 Knapsack problem that **P4** is NP-hard.

In the discrete case where all the parameters and variables are restricted to integers, we can apply the same procedure to cast the Knapsack problem with integer weights to the discrete channel bonding problem. It then follows from the NP-completeness of Knapsack problem with integer weights [23] that the discrete version of **P4** is also NP-complete.

5.7.3 Proof of Lemma 5.1

Consider any feasible scheduling policy mapping to a subset of vertices in $\mathcal{V}$. We first prove that the subset of vertices is an IS of G. Assume by contradiction that

among the subset of vertices there are two neighboring vertices u and v connected by an edge denoted by e. Recall the construction of graph G:

- e cannot be a type 1 edge, otherwise u and v interfere with each other, contradicting with the fact that the time-frequency blocks that are allocated to any pair of interference users do not overlap.

- e cannot be a type 2 edge, otherwise, u and v violate the FIFO model, contradicting with the fact that any feasible scheduling policy does not not violate the FIFO model.

- e cannot be a type 3 edge, otherwise u and v correspond to the same request, contradicting with the fact that each request is executed at most once at any feasible scheduling policy.

The above analysis demonstrates that e cannot exist, thus proving via contradiction that the subset of vertices maps to an IS of G.

We then prove that the parent of any induced vertex in the subset is also in the subset. Based on the definitions of the feasible policy and the adapted FIFO model, for each request instantiated by an induced vertex in the subset, it must reuse the time-frequency block of a request instantiated by an elementary vertex in the subset. It follows from the construction of the request graph that the elementary vertex is the parent of the induced.

By combining the above analysis, each feasible policy maps to a RIS of G.

5.7.4 Proof of Lemma 5.2

Before giving the proof of Lemma 5.2, we first prove the following lemma.

Lemma 5.7. *For any pair of elementary vertices $v_1 \in \mathcal{V}_{k_1}^+$ and $v_2 \in \mathcal{V}_{k_2}^+$, where $b_{k_1} = b_{k_2}$, $l_{v_1} = l_{v_2}$, $h_{v_1} = h_{v_2}$, $\min\{t_{v_1}, t_{v_2}\} \geq \max\{a_{k_1}, a_{k_2}\}$, $k_1, k_2 \in \mathcal{N}$, it holds that at most one of v_1 and v_2 can be chosen at any feasible scheduling policy.*

Proof. Assume, by contradiction, that both v_1 and v_2 are chosen at a feasible scheduling policy. We proceed by distinguishing the following two cases.

- Case 1: $k_1 = k_2$. v_1 and v_2 instantiate the same request, thus are connected by a type 3 edge.

- Case 2: $k_1 \neq k_2$. Without loss of generality, assume $k_1 < k_2$. If $t_{v_1} > t_{v_2}$, it follows from the construction of G that v_1 and v_2 are connected by a type 2 edge. If $t_{v_1} \leq t_{v_2}$, it holds that $k_2 \in \mathcal{N}_{v_1}^+$ because of $t_{v_1} \geq a_{k_2}$ and $t_{v_1} + l_{v_1} - 1 \leq t_{v_2} + l_{v_2} - 1 \leq d_{k_2}$, i.e., the request k_2 fits into the time interval and the spectrum corresponding to the vertex v_1. It then follows from the construction of G that v_1 and v_2 are connected by a type 2 edge.

In both cases, v_1 and v_2 are connected by an edge, contradicting to the fact that any pair of vertices at any feasible scheduling policy is not adjacent to each other. Lemma 5.7 is thus proved. □

Armed with the above lemma, we next prove Lemma 5.2.

Consider a group b and a (t, s) pair. For each slot $\tau \in [\max\{0, t - L_b + 1\}, t]$, let $\mathcal{V}_{b,\tau}^{\alpha}(t, s)$ denote the set of elementary vertices in $\mathcal{V}^{\alpha}(t, s)$ of group b whose left boundaries are slot τ, i.e., $\mathcal{V}_{b,\tau}^{\alpha}(t, s) \triangleq \{u : u \in \mathcal{V}_k^+, \forall k \in \mathcal{N}_b, t_u = \tau, u \text{ is active at } (t, s)\}$. Noticing that the length of the rectangle corresponding to each vertex of group b is at most L_b, the left boundary of any active vertex at (t, s) falls into $[\max\{0, t - L_b + 1\}, t]$. It then follows from the definition of $\mathcal{V}_{b,\tau}^{\alpha}(t, s)$ that the length of rectangle corresponding to each vertex in $\mathcal{V}_{b,\tau}^{\alpha}(t, s)$ is at least $t - \tau + 1$, i.e., $l_v + \tau - 1 \geq t, \forall v \in \mathcal{V}_{b,\tau}^{\alpha}(t, s)$. Let $\mathcal{N}_{b,\tau}^{\alpha}(t, s)$ denote the set of requests having at least one vertex in $\mathcal{V}_{b,\tau}^{\alpha}(t, s)$. It follows from the definitions of $\mathcal{V}_{b,\tau}^{\alpha}(t, s)$ and $\mathcal{N}_{b,\tau}^{\alpha}(t, s)$ that $a_i \leq \tau, \forall i \in \mathcal{N}_{b,\tau}^{\alpha}(t, s)$.

Consider a group b. For each slot $\tau \in [\max\{0, t - L_b + 1\}, t]$ and each (l, f) pair, where $l \in [\max\{t - \tau + 1, l_b\}, L_b]$ and $f \in [f_b, F_b]$, because $t_v = \tau, \forall v \in \mathcal{V}_{b,\tau}^{\alpha}(t, s)$ and $a_i \leq \tau, \forall i \in \mathcal{N}_{b,\tau}^{\alpha}(t, s)$, it then follows from Lemma 5.7 that there is at most one vertex $v \in \mathcal{V}_{b,\tau}^{\alpha}(t, s)$ with $l_v = l$ and $h_v = f$ at any feasible scheduling policy. Therefore, there are at most $(F_b - f_b + 1) \min\{L_b - l_b + 1, L_b - (t - \tau)\}$ elementary vertices in $\mathcal{V}_{b,\tau}^{\alpha}(t, s)$ at any feasible scheduling policy.

By summing over $(F_b - f_b + 1) \min\{L_b - l_b + 1, L_b - (t - \tau)\}$ for all slots $\tau \in [t - L_b + 1, t]$, the size of elementary vertices in $\mathcal{V}^{\alpha}(t, s)$ belonging to group b at any feasible scheduling policy is upper-bounded by

$$\begin{aligned}
&\sum_{\tau=t-L_b+1}^{t} (F_b - f_b + 1) \min\{L_b - l_b + 1, L_b - (t - \tau)\} \\
=&(L_b + l_b)(L_b - l_b + 1)(F_b - f_b + 1)/2.
\end{aligned}$$

We then prove that the vertices in $\mathcal{V}^{\alpha}(t, s)$ at any feasible scheduling policy belong

to a same group. Assume by contradiction that for a (t, s) pair, there is a pair of vertices $u, v \in \mathcal{V}^{\alpha}(t, s)$ belonging to different groups at a feasible scheduling policy. It follows from the construction of G that u and v are connected by a type 1 edge. This contradicts to the fact that any feasible scheduling policy maps to a RIS of G. Therefore, for each (t, s) pair, the size of $\mathcal{V}^{\alpha}(t, s)$ at any feasible scheduling policy is upper-bounded by $\max_{b \in \mathcal{B}^{\alpha}(t,s)}(L_b + l_b)(L_b - l_b + 1)(F_b - f_b + 1)/2$.

5.7.5 Proof of Lemma 5.3

Consider each group $b \in \mathcal{B}^c(i)$ and each slot $\tau \geq \max\{0, a_i + l_{min} - l_{max}\}$. Let $\mathcal{V}^c_{\tau}(i)$ denote the set of elementary vertices $v \in \mathcal{V}^c(i)$ with $t_v = \tau$. Let $\mathcal{N}^c_{\tau}(i)$ denote the set of requests having at least one vertex in $\mathcal{V}^c_{\tau}(i)$. It follows from the definition of $\mathcal{V}^c_{\tau}(i)$ and $\mathcal{N}^c_{\tau}(i)$ that $\max_{k \in \mathcal{N}^c_{\tau}(i)} a_k \leq \tau$.

- For each slot $\tau \in [\max\{0, a_i + l_{min} - l_{max}\}, a_i)$, because $t_v = \tau, \forall v \in \mathcal{V}^c_{\tau}(i)$ and $\max_{k \in \mathcal{N}^c_{\tau}(i)} a_k \leq \tau$, i.e., $t_v \geq \max_{k \in \mathcal{N}^c_{\tau}(i)} a_k, \forall v \in \mathcal{V}^c_{\tau}(i)$, it follows from Lemma 5.7 that for each (l, f) pair, where $l \in [l_{min}, l_{max}]$ and $f \in [f_{min}, f_{max}]$, there is at most one vertex $u \in \mathcal{V}^{\tau}(i)$ at any feasible scheduling policy with $l_u = l$ and $h_u = f$. It then follows from $l \in [l_{min}, l_{max}]$ and $f \in [f_{min}, f_{max}]$ that the size of $\mathcal{V}^c_{\tau}(i)$ at any feasible scheduling policy is upper-bounded by $(l_{max} - l_{min} + 1)(f_{max} - f_{min} + 1)$. Therefore, the size of $\bigcup_{t \in [\max\{0, a_i + l_{min} - l_{max}\}, a_i)} \mathcal{V}^c_t(i)$ at any feasible scheduling policy is upper-bounded by $(l_{max} - l_{min} + 1)(f_{max} - f_{min} + 1)(l_{max} - l_{min})$.

- For the slots $\tau \geq a_i$, it follows from the definition of $\mathcal{V}^c(i)$ that the arrival time of any request having a vertex in $\mathcal{V}^c(i)$ is no later than slot a_i. It then follows from $\bigcup_{\tau \geq a_i} \mathcal{V}^c_{\tau}(i) \subseteq \mathcal{V}^c(i)$ that $a_k \leq a_i, \forall k \in \bigcup_{\tau \geq a_i} \mathcal{N}^c_{\tau}(i)$. Because of $t_v \geq a_i, \forall v \in \mathcal{V}^c_{\tau}(i)$, it follows from Lemma 5.7 that for each (l, f) pair, where $l \in [l_{min}, l_{max}]$ and $f \in [f_{min}, f_{max}]$, there is at most one vertex u in $\bigcup_{\tau \geq a_i} \mathcal{V}^c_{\tau}(i)$ at any feasible scheduling policy with $l_u = l$ and $h_u = f$. Therefore, the size of $\bigcup_{\tau \geq a_i} \mathcal{V}^c_{\tau}(i)$ at any feasible scheduling policy is upper-bounded by $(l_{max} - l_{min} + 1)(f_{max} - f_{min} + 1)$.

By combining the above analysis, for each group b and each request $i \in \mathcal{N}_b$, there are at most $(l_{max} - l_{min} + 1)^2(f_{max} - f_{min} + 1)$ elementary vertices in $\bigcup_{\tau \geq \max\{0, a_i + l_{min} - l_{max}\}} \mathcal{V}^c_{\tau}(i)$ at any feasible scheduling policy. Thus, it then follows from $\mathcal{V}^c(i) = \bigcup_{\tau \geq \max\{0, a_i + l_{min} - l_{max}\}} \mathcal{V}^c_{\tau}(i)$ that the size of elementary vertices in $\mathcal{V}^c(i)$ at any feasible scheduling policy is upper-bounded by $(l_{max} - l_{min} + 1)^2(f_{max} - f_{min} + 1)$.

5.7.6 Proof of Lemma 5.4

We first prove that any elementary vertex in $\mathcal{V}'$ is colored by Algorithm 5. Consider each elementary vertex $v \in \mathcal{V}'$ and the vertex set Γ_v, where let Γ_v denote the set of vertices including v to be colored in the iteration of coloring v. We prove that each vertex in Γ_v is colored by Algorithm 5 in Lemma 5.8. Let Γ_v^* denote the set of vertices from which the vertices in Γ_v are generated.

Lemma 5.8. *Each vertex in Γ_v is colored by Algorithm 5.*

Proof. In the iteration of coloring v, the color that cannot be used to Γ_v have been used to color the vertices that are generated from Γ_v^* and the neighbors of Γ_v^*. The vertices that are generated from Γ_v^* and the neighbors of Γ_v^* can be divided into the following three classes. Let b denote the group to which the request covered by v belongs.

- **The first class of vertices generated from the vertices in $\mathcal{V}$ that are connected with Γ_v^* by type 1 edges.** When we color the vertices in Γ_v, it follows from Step 2 that all elementary vertices $r \in \mathcal{V}'$ with $t_r < t_v$ have been colored and no elementary vertex r with $t_r > t_v$ is colored. Therefore, the colors, which cannot be used to Γ_v, have been used to color the elementary vertices in $\mathcal{V}'$ active at $(t_v, s), \forall s \in [s_v, s_v + h_v - 1]$.

  For each $(t_v, s), s \in [s_v, s_v + h_v - 1]$ pair, it follows from the constraint (5.4) that the number of active elementary vertices in $\mathcal{V}'$ at (t_v, s) is upper-bounded by

  $$\sum_{u \in \mathcal{V}^\alpha(t_v, s)} \lfloor y_v^* \hat{N} \rfloor \leq \sum_{u \in \mathcal{V}^\alpha(t_v, s)} y_v^* \hat{N} \leq D_2 \cdot \hat{N}.$$

  By summing over the number of $D_2 \cdot \hat{N}$ active elementary vertices for all $(t_v, s), s \in [s_v, s_v + h_v - 1]$ pairs, there are at most $h_v \cdot D_2 \cdot \hat{N}$ elementary vertices in $\mathcal{V}'$ including v that are active at $(t_v, s), \forall s \in [s_v, s_v + h_v - 1]$ pairs. Therefore, the number of colors, which cannot be used to Γ_v and which have been used to color all the first class of vertices in the iteration of coloring v, is upper-bounded by $h_v \cdot D_2 \cdot \hat{N} - h_v$. It follows from $h_v \leq f_{max}$ that $f_{max} \cdot D_2 \cdot \hat{N} - f_{max}$.

- **The second class of vertices generated from the vertices in $\mathcal{V}$ that are connected with Γ_v^* by type 2 edges.** For each user $k \in \mathcal{N}$, it then follows

from constraint (5.5) that

$$\sum_{v \in \mathcal{V}^c(k)} \lfloor y_v^* \hat{N} \rfloor \leq \sum_{v \in \mathcal{V}^c(k)} y_v^* \hat{N} \leq D_1 \hat{N}.$$

Therefore, the number of elementary vertices generated from the vertices in $\mathcal{V}^c(k)$ is upper-bounded by $D_1 \hat{N}$.

We first note that the current proof is based on that the vertices in $\mathcal{V}^c(k)$ are adjacent to each other. Clearly, the edges that connect each pair of vertices in $\mathcal{V}^c(k)$ include all type 2 edges between the vertices in $\mathcal{V}^c(k)$.

Consider any pair of requests k_1, k_2 with $k_1 < k_2$ and $b_{k_1} = b_{k_2}$. Based on the construction of G, there is no type 2 edge between any vertex in $\mathcal{V}^c(k_1) \backslash \mathcal{V}^c(k_2)$ and any vertex in $\mathcal{V}^c(k_2) \backslash \mathcal{V}^c(k_1)$. When Algorithm 5 colors the vertices generated from $\mathcal{V}^c(k_2)$, all vertices u generated from $\mathcal{V}^c(k_1) \setminus \mathcal{V}^c(k_2)$ have been colored because of $t_u < \min\{t_\nu : \nu \in \mathcal{V}^c(k_2)\}$. Thus, the colors that are used to the vertices generated from $\mathcal{V}^c(k_1) \setminus \mathcal{V}^c(k_2)$ can be used to the vertices generated from $\mathcal{V}^c(k_2) \setminus \mathcal{V}^c(k_1)$ if there is also no type 1 and 3 edge. Since for each request $k \in \mathcal{N}$, the number of elementary vertices generated from the vertices in $\mathcal{V}^c(k)$ is at most $D_1 \hat{N}$, the number of colors used to the second class of vertices of a same group is upper-bounded by $D_1 \hat{N}$.

- **The third class of vertices generated from Γ_v^* and the vertices in $\mathcal{V}$ that are connected with Γ_v^* by type 3 edges.** For each vertex $u \in \Gamma_v$, let j denote the request covered by u. Based on the definition of $\mathcal{V}^c(j)$, the elementary vertices in $\mathcal{V}_j$ belong to $\mathcal{V}^c(j)$ and the parent of each induced vertex in $\mathcal{V}_j$ belongs to $\mathcal{V}_j$. Thus, any pair of vertices generated from $\mathcal{V}_j$ are not colored by a same color if the vertices in $\mathcal{V}^c(j)$ are adjacent to each other. Because we derive the number of colors that are occupied by the second class of vertices based on that the vertices in $\mathcal{V}^c(k), \forall k \in \mathcal{N}$ are adjacent to each other, when Algorithm 5 colors Γ_v, the colors that have been used to its third class of vertices are considered by the analysis of the second class of vertices.

By combining the above analysis, it holds that the number of colors that have been used to the vertices generated from Γ_v^* and the neighbors of Γ_v^* is at most $(D_1 + f_{max} \cdot D_2)\hat{N} - f_{max} - 1$. Therefore, the vertices in Γ_v is colored by a color in $\mathcal{C}$ because of $|\mathcal{C}| = (D_1 + f_{max} \cdot D_2)\hat{N} - f_{max}$. The lemma is thus proved. □

It then follows from Algorithm 5 and Lemma 5.8 that all elementary vertices in $\mathcal{V}'$ are colored.

We now prove that all induced vertices are also colored by Algorithm 5. Because each induced vertex in $\mathcal{V}$ has only one parent, we just need to prove that for each elementary vertex $v \in \mathcal{V}$ and each request $k \in \mathcal{N}_v^+$, when the vertices generated from v are colored, all vertices generated from the induced vertices in $\mathcal{V}_k$ whose parents are v are also colored.

Consider each elementary vertex $v \in \mathcal{V}$ and each request $k \in \mathcal{N}_v^+$. It follows from the constraint (5.3) that

$$\sum_{\nu \in \mathcal{V}_k, v \prec \nu} \lfloor y_\nu^* \hat{N} \rfloor \leq \lfloor y_v^* \hat{N} \rfloor.$$

Thus, the number of vertices in $\mathcal{V}'_{k,v}$ is upper-bounded by the number of vertices generated from v. When Algorithm 5 colors an vertex v' generated from v, it follows from Step 2 of Algorithm 5 that there is a vertex in $\mathcal{V}'_{k,v}$ is also colored using the same color as v' if there is at least one uncolored vertex in $\mathcal{V}'_{k,v}$. Therefore, when all elementary vertices generated from v are colored, all induced vertices in $\mathcal{V}'_{k,v}$ are also colored.

By combining the above analysis, it holds that all vertices in $\mathcal{V}'$ are colored by Algorithm 5. Lemma 5.4 is thus proved.

5.7.7 Proof of Lemma 5.5

We first prove that the set vertices, from which the vertices colored by c are generated, is not adjacent to each other. For any elementary vertex $v \in \mathcal{V}'$ colored by c, there is only one elementary vertex in Γ_v, and no two of vertices cover a same request. Based on the construction of G, any pair of vertices in Γ_v^* is not adjacent to each other. It follows from Algorithm 5 that c is not used to color any other vertex generated from Γ_v^* and the neighbors of Γ_v^*. Therefore, any pair of vertices colored by c is not generated from a same vertex and any pair of adjacent vertices, indicating that the set of vertices, from which the vertices colored by c are generated, satisfies the first property of RIS.

For any induced vertex $u' \in \mathcal{V}'$ colored by the color c, let u denote the vertex from which u' is generated. By the proof of Lemma 5.4, their is a vertex, which is generated from the parent of u, colored by c, indicating that the set of vertices, from which the vertices colored by c are generated, satisfies the second property for RIS of G.

By combining the above analysis, it holds that the set of vertices, from which the

vertices colored by c are generated, induces a RIS. Lemma 5.5 is thus proved.

5.7.8 Proof of Lemma 5.6

It follows from Lemma 5.5 that the set of vertices, denoted by $\hat{\mathcal{V}}$, output by Algorithm 5 is a RIS. For each vertex $v \in \hat{\mathcal{V}}$, let i denote the user corresponding to v.

- Case 1: v is an elementary vertex. There is no elementary vertex $u \in \hat{\mathcal{V}}$ belonging to group b_i with $\beta_u < \beta_v$ such that i can fit into the time-frequency block of u. Assume, by contradiction, that there is an elementary vertex $u \in \hat{\mathcal{V}}$ of group b_i with $\beta_u < \beta_v$ whose corresponding time-frequency block fits for serving i. If $t_u > t_v$, u and v are connected by a type 2 edge because of $\beta_u < \beta_v$; if $t_u \leq t_v$, u and v are connected by a type 2 edge because i fits into the time-frequency block of u, i.e., $i \in \mathcal{N}_u^+$. In both case, u and v are connected by an edge, contradicting to the fact that $\hat{\mathcal{V}}$ is a RIS.

- Case 2: v is an induced vertex. There is no elementary vertex $u \in \hat{\mathcal{V}}$ belonging to the same group as i with $\beta_u < \beta_v$ such that i can fit into the time-frequency block of u; otherwise, it follows from Step 3 that v has been replaced by an induced vertex in $\mathcal{V}_i$ whose parent has the highest priority among the elementary vertices $\mu \in \hat{\mathcal{V}}$ belonging to group b_i with $i \in \mathcal{N}_\mu^+$.

Therefore, the set of vertices output by Algorithm 5 maps to a feasible scheduling policy.

5.7.9 Proof of Theorem 5.2

Denote the utility of the optimal feasible scheduling policy by OPT. Let $A \triangleq (D_1 + f_{max} \cdot D_2)$. We have shown that each color induces a RIS of G in Lemma 5.5. It follows from Lemma 5.4 that the coloring of $\mathcal{V}'$ uses at most $A\hat{N} - f_{max}$ colors. We thus get at most $A\hat{N} - f_{max}$ RISes, denoted by $\hat{\mathcal{V}}_i, 1 \leq i \leq I$, where $I \leq A\hat{N}$. Let $\mathcal{I}^*$ denote the RIS output by Algorithm 5. Since each vertex v in G is replaced by $\lfloor y_v^* \hat{N} \rfloor$ new vertices, each of which is colored by Algorithm 5, we have

$$\begin{aligned}\sum_{v \in \mathcal{I}^*} w_v &\geq \frac{\hat{N}}{|\mathcal{C}|}\left(\sum_{v \in \mathcal{V}} w_v y_v^* - \sum_{v \in \mathcal{V}} \frac{w_v}{\hat{N}}\right) \\ &> \frac{1}{A}\left(OPT - \frac{1}{\delta^\epsilon} \max_{i \in \mathcal{N}} \max_{v \in \mathcal{V}_i^+} (w_v + \sum_{j \in \mathcal{N}_v^+} \sum_{r \in \mathcal{V}_j, v \prec r} w_r)\right)\end{aligned}$$

where the second inequality follows that the value of an optimal fractional LP solution is an upper bound on the value of the feasible scheduling policy and the total value decreased by Step 1 is upper-bounded by

$$\begin{aligned}\sum_{v\in\mathcal{V}}\frac{w_v}{\hat{N}} &= \frac{1}{(F-f_{min}+1)\delta^{1+\epsilon}}\sum_{i\in\mathcal{N}}\sum_{v\in\mathcal{V}_i^+}(w_v+\sum_{k\in\mathcal{N}_v^+}\sum_{r\in\mathcal{V}_k,v\prec r}w_r)\\ &\leq \frac{(F-f_{min}+1)\delta}{(F-f_{min}+1)\delta^{1+\epsilon}}\max_{i\in\mathcal{N}}\max_{v\in\mathcal{V}_i^+}(w_v+\sum_{j\in\mathcal{N}_v^+}\sum_{r\in\mathcal{V}_j,v\prec r}w_r)\\ &\leq \frac{1}{\delta^{\epsilon}}\max_{i\in\mathcal{N}}\max_{v\in\mathcal{V}_i^+}(w_v+\sum_{j\in\mathcal{N}_v^+}\sum_{r\in\mathcal{V}_j,v\prec r}w_r),\end{aligned}$$

where the equal follows from that each induced vertex has only one parent, and the first inequality follows from that there are at most $(F-f_{min}+1)\delta$ elementary vertices in $\mathcal{V}$.

As it holds that

$$\max_{i\in\mathcal{N}}\max_{v\in\mathcal{V}_i^+}(w_v+\sum_{j\in\mathcal{N}_v^+}\sum_{r\in\mathcal{V}_j,v\prec r}w_r)\leq OPT,$$

we then have

$$\sum_{v\in\mathcal{I}^*}w_v>\frac{1}{A}(OPT-\frac{1}{\delta^{\epsilon}}\cdot OPT),$$

The theorem is thus proved.

5.7.10 Proof of Theorem 5.4

For a slot $t\in\mathcal{T}$. Let $\mathcal{O}$ denote the set of requests that are served by the optimum scheduling policy.

We now prove that each request in $\mathcal{O}$ is ready at a slot in $\mathcal{T}$. Consider each request $k\in\mathcal{O}$. If $a_k\in\mathcal{T}$, it follows from the definition of ready request that k is ready at slot a_k, i.e., $k\in\Omega_{a_k}$; if $a_k\notin\mathcal{T}$, let $t=\max\{\tau:\tau\in\mathcal{T},\tau<a_k\}$ and $l^*=\max_{k\in\mathcal{N}_t^*}l_k$. We prove that request k is ready at slot $t+l^*$ and $t+l^*\in\mathcal{T}$. We prove it in the following three steps.

- **Step 1:** We prove that $t<a_k<t+l^*$. Assume, by contradiction, that $a_k\geq t+l^*$. It follows from $t=\max\{\tau:\tau\in\mathcal{T},\tau<a_k\}$ and $a_k\notin\mathcal{T}$ that each slot in $[t+l^*,a_k]$ is not occupied when Algorithm 6 considers it. Because request k is ready at slot a_k, Algorithm 6 starts serving at least one request at slot a_k, i.e., $a_k\in\mathcal{T}$, contradicting with $a_k\notin\mathcal{T}$. It then holds that $t<a_k<t+l^*$.

- **Step 2:** We prove that request k is ready at slot $t + l^*$. Since the slackness for each request i is at least $l_{max} + l_i - 1$, we can derive that $a_k < t + l^* \leq d_k - l_k + 1$ because of $t < a_k < t + l^*$ and $l^* \leq l_{max}$. It also holds that starting serving request k at slot $t + l^*$ does not violate the FIFO model because of $a_j \leq t, \forall j \in \mathcal{N}_t^*$ and $a_k > t$. Therefore, request k is ready at slot $t + l^*$, i.e., $k \in \Omega_{t+l^*}$.

  Step 3: We prove that $t+l^* \in \mathcal{T}$. Because requests in $\mathcal{N}_t^*$ are served completely at slot $t+l^*-1$, slot $t+l^*$ is not occupied when Algorithm 6 considers it. Therefore, Algorithm 6 starts serving at least one request at slot $t + l^*$ since request k is ready at slot $t + l^*$, i.e., $t + l^* \in \mathcal{T}$.

Based on the above analysis, we can derive that the total reward of the optimum scheduling policy can be upper-bounded by $\sum_{\tau \in \mathcal{T}} \sum_{i \in \Omega_t} w_i$, i.e., $\sum_{i \in \mathcal{O}} w_i \leq \sum_{\tau \in \mathcal{T}} \sum_{i \in \Omega_t} w_i$.

Consider each slot $t \in \mathcal{T}$. Because each request i satisfies $f_i \leq F$, in the first iteration of slot t Algorithm 6 starts serving all requests in Ω_t of group b^* at slot t, i.e., $\{k : k \in \Omega_t, b_k = b^*\} \in \mathcal{N}_t^*$, where $b^* = \text{argmax}_{b \in \mathcal{B}(\Omega_t)} \sum_{k \in \Omega_t, b_k = b} w_k$, the total reward of requests in Ω_t is

$$\begin{aligned}\sum_{i \in \Omega_t} w_i &\leq \sum_{b \in \mathcal{B}(\Omega_t)} \sum_{k \in \Omega_t, b_k = b} w_k \\ &\leq |\mathcal{B}(\Omega_t)| \cdot \sum_{k \in \Omega_t, b_k = b^*} w_k \\ &\leq |\mathcal{B}(\Omega_t)| \cdot \sum_{i \in \mathcal{N}_t^*} w_i.\end{aligned}$$

Therefore, the total reward of the optimum scheduling policy can be upper-bounded by

$$\sum_{i \in \mathcal{O}} w_i \leq \sum_{t \in \mathcal{T}} \sum_{i \in \Omega_t} w_i \leq \sum_{t \in \mathcal{T}} |\mathcal{B}(\Omega_t)| \sum_{i \in \mathcal{N}_t^*} w_i \leq B_{max} \cdot \sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{N}_t^*} w_i.$$

It follows from the definitions of $\mathcal{T}$ and $\mathcal{N}_t^*$ that the utility of Algorithm 6 is at least $\sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{N}_t^*} w_i$. Hence, Algorithm 6 has B_{max} competitive ratio if the slackness for each request i is at least $l_{max} + l_i - 1$.

Chapter 6

Conclusion and Prospective

6.1 Thesis Summary

This thesis has dedicated to the fundamental batching task scheduling problems, each of which has its own particularities and calls for specific analysis that cannot draw upon existing results, at the theoretical modeling and analysis and the approximation algorithm design, with Chapter 2 reviewing the related literature, Chapter 3 focusing on developing an algorithmic framework for the baseline scenario of batching task scheduling problem, Chapters 4 and 5 presenting approximation algorithms in both offline and online settings with mathematically proven performance guarantee for the problems of downlink transmission scheduling with data sharing and contiguous-resource batching task scheduling, respectively. More specifically, Chapter 3 has developed an algorithmic framework achieving 1/2-optimality for the baseline scenario of batching task scheduling problem, outperforming the best known result, and derived the first deterministic approximation algorithm outputting a $l_{min}/(2(l_{min}+1))$-optimal scheduling policy for the generalized proportional broadcast problem by applying our algorithmic framework to the problem. In Chapters 4 and 5, we have formulated and analyzed the downlink transmission scheduling with data sharing and contiguous-resource batching task scheduling problems, which significantly generalize the baseline scenario of batching task scheduling problem, respectively, and we have studied the problems in both offline and online cases by establishing the problem hardness and developing deterministic approximation algorithms with mathematically proven performance guarantee. We have conducted numeric experiments under a variety of typical parameter settings to demonstrate the effectiveness of our algorithms for the above problems.

By summarizing the previous problems, the core technicality in our design is an LP

relaxation mechanism and a rounding and coloring approach that turns the solution of the LP relaxation to a feasible scheduling policy in offline case. In the online case, at each slot, at which no request or task is being executed, our online algorithms start serving a set of users requesting the same packet or belonging the same group such that the total reward of the users is maximum.

6.2 Open Questions and Future Work

In this section, we develop the discussion on open issues and questions and future work. The first is to extend the problems we address in Chapters 4 and 5 to the setting of multiple resource pools, where each task is to be scheduled in one of them, thus adding another dimension to the problems. Note that our algorithmic framework can be easily extended to the setting of multiple resources. The second and third future directions we expect to look into are to study *dependent batching task scheduling* and to concern *flexible FIFO model* in the problems of downlink transmission scheduling with data sharing and contiguous-resource batching task scheduling.

6.2.1 Dependent batching task scheduling

Throughout the full text, the tasks in the problems we address are independent. We expect to look into a generic task scheduling problem: a set of *interdependent* tasks need to be executed, each associated with a time window and, if admitted, needs to be executed within the window by exclusively using the resource; some tasks can be executed simultaneously by sharing the resource, while others require exclusive use of resource. The dependency among tasks forms a directed acyclic graph such that each task can be executed iff all the tasks preceding it are completely executed. The goal is to seek an optimum scheduling algorithm maximizing the overall system utility. The above scheduling problem arises in a variety of engineering fields where computing, communication, and storage resources are potential bottleneck and thus need to be carefully scheduled.

Despite its theoretical and practical importance, the problem of scheduling interdependent batching tasks is still a largely unexplored area, where the challenges brought by task dependency and batching execution need to be addressed holistically in the design of scheduling algorithms maximizing the system overall utility. Thus, we expect that the problems not only consider task batching, i.e., some tasks can be executed

simultaneously, but also task dependency, i.e., the dependency among tasks forms a directed acyclic graph. This adds an even stronger combinatorial flavor to the problem and deserves a careful investigation.

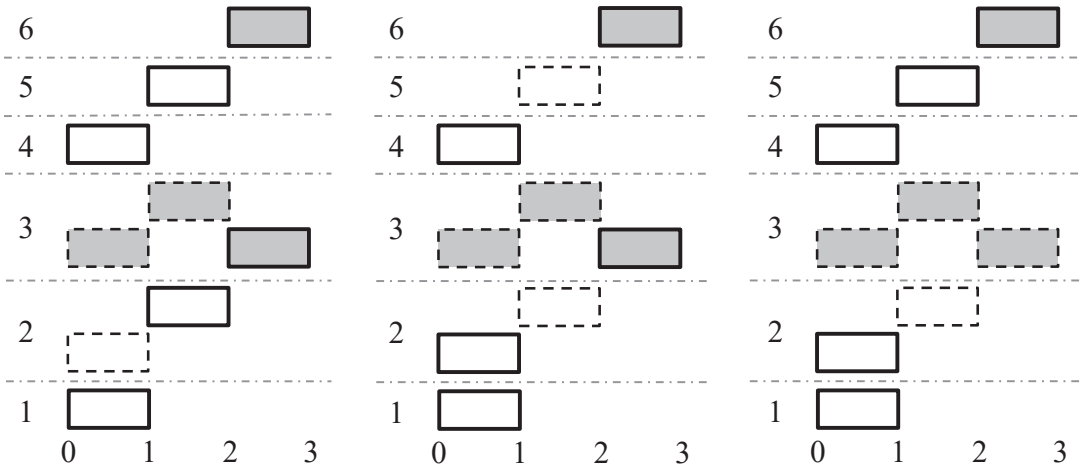
6.2.2 Flexible FIFO model

Our works presented in Chapters 3 and 4 focus on the adapted FIFO model illustrated in Figures 4.1 and 5.2, where when any scheduling policy serves user i, any user $j > i$ that requests the packet b_i or belongs to group b_i must be batched with request i, as long as j can fit into the time interval or time-frequency block of i. In many practical scenarios, admitted requests may be served according to a more flexible FIFO model.

Consider the contiguous-resource batching task scheduling problem. We now consider a flexible FIFO model defined as below.

Definition 6.1 (Flexible FIFO model)**.** *When any scheduling policy serves request i, any request $j > i$ that belongs to group b_i can but does not have to be served simultaneously with request i if j can fit in the service time of i and the spectrum allocated to i.*

Note that the flexible FIFO model for the problem of downlink transmission scheduling with data sharing is similar.



(a) Flexible FIFO model (b) Adapted FIFO model (c) Standard FIFO model

Figure 6.1: Illustration of the flexible FIFO model vs. the adapted and standard FIFO models

Example 6.1. *To illustrate the flexible FIFO model against the adapted and standard FIFO models, we consider an example composed of six requests, indexed from 1 to*

6, with the corresponding request quadruples $(0, 0, F, 1), (0, 1, F, 1), (0, 2, F, 1), (0, 0, F, 1)$ *and* $(1, 1, F, 1), (2, 2, F, 1)$*, respectively. Requests 1,2,4 and 5 belong to group 1, and requests 3,6 belongs to group 2. The reward of each request is 1. The rectangles in Figure 6.1 illustrate the possible executions of the requests, with the height being the quantity of requested spectrum and the length being the requested execution time.*

- *Figure 6.1(a) illustrates the optimal scheduling policy under the flexible FIFO model, which serves requests 1 and 4 at slot* 0*, requests 2 and 5 at slot 1, requests 3 and 6 at slot 3. Request 2 is not served simultaneously with request 1 at slot 0 in order to serve request 5 at slot 1 under a higher priority.*

- *Figure 6.1(b) illustrates the optimal scheduling policy under the adapted FIFO model, which serves requests 1,2 and 4 at slot 0, requests 3 and 6 at slot 2, or requests 1,2 and 4 at slot 0, request 5 at slot 1, request 6 at slot 2. Different from the adapted FIFO model, request 2 must be served simultaneously with request 1.*

- *Figure 6.1(c) illustrates the optimal scheduling policy under the standard FIFO model, which serves requests 1,2 and 4 at slot 0, request 5 at slot 1, request 6 at slot 2. Different from the flexible and adapted FIFO models, any admitted request should be started no later than any admitted request arriving later.*

We can check that under flexible FIFO policy, all the six requests are served; under the adapted and standard FIFO models, at most five requests can be served.

The flexible FIFO model is more flexible than the adapted and standard FIFO models and leads to better efficiency. The flexible FIFO model is technically more involved than the adapted and standard FIFO models. Any mathematical framework under the flexible FIFO model can be extended to the adapted and standard FIFO models.

Based on the definition of flexible FIFO model, we expect to seek the optimal scheduling policy maximizing the overall system utility for the problems of downlink transmission scheduling with data sharing and contiguous-resource batching task scheduling. We also expect to seek the optimal scheduling policy minimizing the total delay cost or minimizing makespan for the problems.

Bibliography

[1] R. Agrawal, A. Bedekar, R. J. La, and V. Subramanian. Class and channel condition based weighted proportional fair scheduler. In *Teletraffic Engineering in the Internet Era*, volume 4, pages 553 – 567. Elsevier, 2001.

[2] R. Agrawal and V. Subramanian. Optimality of certain channel aware scheduling policies. In *Proc. Allerton*, 2002.

[3] M. Andrews, K. Kumaran, K. Ramanan, A. Stolyar, P. Whiting, and R. Vijayakumar. Providing quality of service over a shared wireless link. *IEEE Communications Magazine*, 39(2):150–154, 2001.

[4] N. Bansal, D. Coppersmith, and M. Sviridenko. Improved approximation algorithms for broadcast scheduling. *SIAM Journal on Computing*, 38(3):1157–1174, 2008.

[5] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. S. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of ACM*, 48(5):1069–1090, Sept. 2001.

[6] A. Bar-Noy, S. Guha, Y. Katz, J. S. Naor, B. Schieber, and H. Shachnai. Throughput maximization of real-time scheduling with batching. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 742–751, Philadelphia, PA, USA, 2002.

[7] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the throughput of multiple machines in real-time scheduling. *SIAM Journal on Computing*, 31(2):331–352, 2001.

[8] R. Bar-Yehuda, M. Beder, and Y. Cohen. Approximation algorithms for bandwidth and storage allocation. *Technical report*, 2005.

[9] R. Bar-Yehuda, M. Beder, Y. Cohen, and D. Rawitz. Resource allocation in bounded degree trees. In *Proc. ESA*, 2006.

[10] P. Berman and B. Dasgupta. Multi-phase algorithms for throughput maximization for real-time scheduling. *Journal of Combinatorial Optimization*, 4(3):307–323, Sep 2000.

[11] G. Calinescu, A. Chakrabarti, H. Karloff, and Y. Rabani. Improved approximation algorithms for resource allocation. In *Proc. IPCO*, 2002.

[12] C. Chekuri, A. Gal, S. Im, S. Khuller, J. Li, R. McCutchen, B. Moseley, and L. Raschid. New models and algorithms for throughput maximization in broadcast scheduling. In *Approximation and Online Algorithms*, pages 71–82. Springer, 2011.

[13] B. Chen, R. Hassin, and M. Tzur. Allocation of bandwidth and storage. *IIE Transactions*, 34(5):501–507, 2002.

[14] J. Chuzhoy, R. Ostrovsky, and Y. Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. *Mathematics of Operations Research*, 31(4):730–738, 2006.

[15] A. Darmann, U. Pferschy, and J. Schauer. Resource allocation with time intervals. *Theoretical Computer Science*, 411(49):4217 – 4234, 2010.

[16] M. Eisen, C. Zhang, L. F. O. Chamon, D. D. Lee, and A. Ribeiro. Learning optimal resource allocations in wireless systems. *IEEE Transactions on Signal Processing*, 67(10):2775–2790, May 2019.

[17] S. P. Y. Fung, F. Y. L. Chin, and C. K. Poon. Laxity helps in broadcast scheduling. In *In Proc. 9th Italian Conference on Theoretical Computer Science, LNCS 3701*, pages 251–264, 2005.

[18] R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan. Dependent rounding and its applications to approximation algorithms. *Journal of ACM*, 53(3):324–360, May 2006.

[19] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1(3):237 – 267, 1976.

[20] E. Gelenbe, Xiaowen Mang, and R. Onvural. Bandwidth allocation and call admission control in high-speed networks. *IEEE Communications Magazine*, 35(5):122–129, 1997.

[21] H. Hoogeveen, C. N. Potts, and G. J. Woeginger. On-line scheduling on a single machine: maximizing the number of early jobs. *Operations Research Letters*, 27(5):193 – 197, 2000.

[22] J. Huang, V. G. Subramanian, R. Agrawal, and R. A. Berry. Downlink scheduling and resource allocation for OFDM systems. *IEEE Transactions on Wireless Communications*, 8(1):288–296, 2009.

[23] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of ACM*, 22(4):463–468, Oct. 1975.

[24] S. Im and M. Sviridenko. New approximations for broadcast scheduling via variants of α-point rounding. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, page 1050–1069, USA, 2015.

[25] S. A. Javadi. Analytical approaches for dynamic scheduling in cloud environments. *SIGMETRICS Perform. Eval. Rev.*, 47(3):14–16, Jan. 2020.

[26] R. M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer, 1972.

[27] E. L. Lawler, J. K. Lenstra, A. H. Rinnooy Kan, and D. B. Shmoys. Chapter 9 sequencing and scheduling: Algorithms and complexity. In *Logistics of Production and Inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, pages 445 – 522. Elsevier, 1993.

[28] S. Leonardi, A. Marchetti-Spaccamela, and A. Vitaletti. Approximation algorithms for bandwidth and storage allocation problems under real time constraints. In *Foundations of Software Technology and Theoretical Computer Science*, pages 409–420. Springer, 2000.

[29] Z. Li and M. Ierapetritou. Process scheduling under uncertainty: Review and challenges. *Computers & Chemical Engineering*, 32(4):715 – 727, 2008. Festschrift devoted to Rex Reklaitis on his 65th Birthday.

[30] J. Mo and J. Walrand. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking*, 8(5):556–567, 2000.

[31] T. Mömke and A. Wiese. A $(2+\epsilon)$-approximation algorithm for the storage allocation problem. In *Proc. ICALP*, 2015.

[32] M. J. Neely. Delay-based network utility maximization. *IEEE/ACM Transactions on Networking*, 21(1):41–54, Feb 2013.

[33] C. A. Phillips, R. N. Uma, and J. Wein. Off-line admission control for general scheduling problems. In *Proc. SODA*, 2000.

[34] H. A. M. Ramli, R. Basukala, K. Sandrasegaran, and R. Patachaianand. Performance of well known packet scheduling algorithms in the downlink 3GPP LTE system. In *Malaysia International Conference on Communications (MICC)*, pages 815–820, 2009.

[35] J. Rubio, A. Pascual-Iserte, D. P. Palomar, and A. Goldsmith. Joint optimization of power and data transfer in multiuser mimo systems. *IEEE Transactions on Signal Processing*, 65(1):212–227, 2017.

[36] S. Ryu, B. Ryu, H. Seo, and M. Shin. Urgency and efficiency based packet scheduling algorithm for OFDMA wireless system. In *Proc. ICC*, volume 4, pages 2779–2785 Vol. 4, 2005.

[37] S. Sakai, M. Togasaki, and K. Yamazaki. A note on greedy algorithms for the maximum weighted independent set problem. *Discrete Applied Mathematics*, 126(2):313 – 322, 2003.

[38] K. Sandrasegaran, H. A. Mohd Ramli, and R. Basukala. Delay-prioritized scheduling (dps) for real time traffic in 3GPP LTE system. In *Proc. IEEE WCNC*, pages 1–6, 2010.

[39] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics. Springer, 2003.

[40] H. Shachnai, A. Voloshin, and S. Zaks. Flexible bandwidth assignment with application to optical networks. *Journal of Scheduling*, 21(3):327–336, 2018.

[41] G. Song, Y. Li, L. J. Cimini, and H. Zheng. Joint channel-aware and queue-aware data scheduling in multiple shared wireless channels. In *Proc. IEEE WCNC*, volume 3, pages 1939–1944, 2004.

[42] F. Spieksma. On the approximability of an interval scheduling problem. *Journal of Scheduling*, 2:215–227, 01 1999.

[43] A. L. Stolyar. On the asymptotic optimality of the gradient scheduling algorithm for multiuser throughput allocation. *Operations Research*, 53(1):12–25, 2005.

[44] L. Thiele. Resource constrained scheduling of uniform algorithm. In *Proc. International Conference on Application Specific Array Processors*, pages 29 – 40, Los Alamitos, CA, USA, 1993.

[45] B. Tian, J. Huang, B. Mozafari, and G. Schoenebeck. Contention-aware lock scheduling for transactional databases. *Proc. VLDB Endow.*, 11(5):648–662, Jan. 2018.

[46] H. Van Dyke Parunak. Characterizing the manufacturing scheduling problem. *Journal of Manufacturing Systems*, 10(3):241 – 259, 1991.

[47] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2010.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre: Planification des tâches de calcul et de communication par lots: base théorique et conception d'algorithmes

Mots clés: Algorithmes d'approximation, Traitement par lots, Ordonnancement

Résumé: Dans cette thèse, nous formulons et analysons une classe de problèmes fondamentaux d'ordonnancement des tâches découlant d'une variété de systèmes informatiques et de communication émergents : les tâches sont divisées en groupes; celles d'un groupe peuvent être mises en lots et exécutées simultanément; l'objectif de l'ordonnanceur est de concevoir des algorithmes d'ordonnancement maximisant l'utilité globale du système. Sous le parapluie générique ci-dessus, nous étudions différentes classes de problèmes d'ordonnancement de tâches par lots, en établissant le cadre théorique correspondant, en concevant des algorithmes d'ordonnancement hors ligne et en ligne, et en illustrant leur application dans la planification des tâches de communication et d'informatique.

Nous commençons par le scénario de base de l'ordonnancement des tâches par lots. Il y a un ensemble de tâches à exécuter sur un certain nombre de machines. Certaines tâches peuvent être exécutées simultanément sur une seule machine, tandis que d'autres nécessitent l'utilisation exclusive d'une machine entière. Nous recherchons une politique d'ordonnancement optimale pour maximiser l'utilité globale du système. Nous développons un cadre algorithmique pour le problème d'ordonnancement ci-dessus dans la forme générique qui peut atteindre 1/2-optimality, surperformant le résultat le plus connu. Nous démontrons ensuite l'application de notre cadre algorithmique pour résoudre le problème de diffusion proportionnelle généralisée en développant le premier algorithme d'approximation déterministe.

Nous formulons et analysons ensuite un problème fondamental de programmation de transmission en liaison descendante dans les systèmes de communication sans fil, composé d'une station de base et d'un ensemble d'utilisateurs, chacun demandant un paquet à servir dans une fenêtre de temps. Certains paquets sont demandés par plusieurs utilisateurs et peuvent être servis simultanément. Dans le problème, chaque demande peut être servie par un sous-ensemble de stratégies de transmission, et les demandes doivent être servies de la manière FIFO. Nous recherchons un algorithme de programmation de transmission en liaison descendante maximisant l'utilité globale du système. Nous établissons d'abord sa dureté, puis développons des algorithmes d'approximation avec une garantie de performance mathématiquement prouvée en termes d'approximation et de rapports compétitifs pour les paramètres hors ligne et en ligne, respectivement.

La troisième contribution concerne l'ordonnancement des tâches de mise en lots des ressources contiguës. Un ensemble de tâches doit être exécuté sur un pool de ressources continues, chacune nécessitant un certain temps et une ressource contiguë; certaines tâches peuvent être exécutées simultanément en lot en partageant la ressource, tandis que d'autres nécessitent une utilisation exclusive de la ressource; les tâches sont servies de la manière FIFO. Nous recherchons une allocation optimale des ressources et la politique d'ordonnancement connexe maximisant l'utilité globale du système. Nous établissons la dureté du problème et développons des algorithmes de programmation d'approximation pour les paramètres hors ligne et en ligne.

Maison du doctorat de l'Université Paris-Saclay
2ème étage aile ouest, Ecole normale supérieure Paris-Saclay
4 avenue des Sciences,
91190 Gif sur Yvette, France

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Title: Scheduling Batching Computing and Communication Tasks: Theoretical Foundation and Algorithm Design

Keywords: Approximation Algorithm, Batching, Task Scheduling

Abstract: In this thesis we formulate and analyze a class of fundamental task scheduling problems arising from a variety of emerging computing and communication systems: tasks are partitioned into groups; those in a group can be batched and executed simultaneously; the goal faced by the scheduler is to design scheduling algorithms maximizing the overall system utility. Under the above generic umbrella, we investigate different classes of batching task scheduling problems, establishing the corresponding theoretical framework, designing both offline and online scheduling algorithms, and illustrating their application in scheduling communication and computing tasks.

We start by the baseline scenario of batching task scheduling. There is a set of tasks to be executed on a number of machines. Some tasks can be executed simultaneously on a single machine, while others require exclusive use of an entire machine. We seek an optimal scheduling policy to maximize the overall system utility. We develop an algorithmic framework for the above scheduling problem in the generic form that can achieve 1/2-optimality, outperforming the best known result. We then demonstrate the application of our algorithmic framework to solve the generalized proportional broadcast problem by developing the first deterministic approximation algorithm.

We then formulate and analyze a fundamental downlink transmission scheduling problem in wireless communication systems, composed of a base station and a set of users, each requesting a packet to be served within a time window. Some packets are requested by several users and can be served simultaneously. In the problem, each request can be served by a subset of transmission strategies, and requests need to be served in the FIFO manner. We seek a downlink transmission scheduling algorithm maximizing the overall system utility. We first establish its hardness, and then develop approximation algorithms with mathematically proven performance guarantee in terms of approximation and competitive ratios for the offline and online settings, respectively.

The third contribution concerns the contiguous-resource batching task scheduling. A set of tasks need to be executed on a pool of continuous resource, each requiring a certain amount of time and contiguous resource; some tasks can be executed simultaneously in batch by sharing the resource, while others requiring exclusive use of the resource; tasks are served in the FIFO manner. We seek an optimal resource allocation and the related scheduling policy maximizing the overall system utility. We establish the hardness of the problem and developing approximation scheduling algorithms for both offline and online settings.

Maison du doctorat de l'Université Paris-Saclay
2ème étage aile ouest, Ecole normale supérieure Paris-Saclay
4 avenue des Sciences,
91190 Gif sur Yvette, France