April 2019

Approximation Algorithms for Problems in Makespan Minimization on Unrelated Parallel Machines

Daniel R. Page
The University of Western Ontario

Supervisor
Solis-Oba, Roberto
The University of Western Ontario

Graduate Program in Computer Science

A thesis submitted in partial fulfillment of the requirements for the degree in Doctor of Philosophy

Abstract

A fundamental problem in scheduling is makespan minimization on unrelated parallel machines ($R||C_{max}$). Let there be a set J of jobs and a set M of parallel machines, where every job $J_j \in J$ has processing time or length $p_{i,j} \in \mathbb{Q}^+$ on machine $M_i \in M$. The goal in $R||C_{max}$ is to schedule the jobs non-preemptively on the machines so as to minimize the length of the schedule, the makespan. A *ρ-approximation algorithm* produces in polynomial time a feasible solution such that its objective value is within a multiplicative factor ρ of the optimum, where ρ is called its approximation ratio. The best-known approximation algorithms for $R||C_{max}$ have approximation ratio 2, but there is no ρ-approximation algorithm with $\rho < 3/2$ for $R||C_{max}$ unless $\mathsf{P} = \mathsf{NP}$. A longstanding open problem in approximation algorithms is to reconcile this hardness gap. We take a two-pronged approach to learn more about the hardness gap of $R||C_{max}$: (1) find approximation algorithms for special cases of $R||C_{max}$ whose approximation ratios are tight (unless $\mathsf{P} = \mathsf{NP}$); (2) identify special cases of $R||C_{max}$ that have the same $3/2$-hardness bound of $R||C_{max}$, but where the approximation barrier of 2 can be broken.

This thesis is divided into four parts. The first two parts investigate a special case of $R||C_{max}$ called the graph balancing problem when every job has one of two lengths and the machines may have one of two speeds. First, we present $3/2$-approximation algorithms for the graph balancing problem with one speed and two job lengths. In the second part of this thesis we give an approximation algorithm for the graph balancing problem with two speeds and two job lengths with approximation ratio $(\sqrt{65} + 7)/8 \approx 1.88278$. In the third part of the thesis we present approximation algorithms and hardness of approximation results for two problems called $R||C_{max}$ with simple job-intersection structure and $R||C_{max}$ with bounded job assignments. We conclude this thesis by presenting algorithmic and computational complexity results for a generalization of $R||C_{max}$ where J is partitioned into sets called bags, and it must be that no two jobs belonging to the same bag are scheduled on the same machine.

Keywords: Theoretical Computer Science, Approximation Algorithms, Scheduling Theory, Makespan Minimization, Unrelated Parallel Machines, Restricted Assignment Problem, Graph Balancing Problem, Job-Intersection Graph, Bounded Job Assignments, Bag Constraints

Co-Authorship Statement

I would like to acknowledge Dr. Roberto Solis-Oba, my supervisor as a co-author of the original articles that form Chapters 3–6. I must also acknowledge Marten Maack as a co-author for some of the results in Chapter 5: Marten Maack made some suggestions to help simplify the algorithm in Chapter 5.2, provided comments on the paper, and proposed the idea for the reduction given in Chapter 5.1.

Acknowledgements

First, I must thank my supervisor Dr. Roberto Solis-Oba. His guidance, insightful ideas, detailed criticism, and honest feedback were helpful in sharpening my skills as a researcher and as a writer. I must also thank him for giving me the trust to explore my research interests in a fairly independent manner, and patiently reviewing all the drafts I would slide under his office door at odd hours of the night sometimes. Thank you so much Roberto for all your help and for giving me the opportunity to work with you.

I would like to thank Marten Maack. Marten had shared my office from late 2015 until mid 2016. He came all the way from Kiel, Germany to work with us as a part of his studies. Thank you again for the conversations and the opportunity to think about problems together.

Next, I must thank the people of the Department of Computer Science at the University of Western Ontario. The department has been nothing but friendly to me, and is full of people that I would be happy to sit down and have a chat with. While I can say I have made plenty of wonderful acquantices among my peers, I must explicitly name two people that I cannot thank enough for the time they gave me and would happily call my friends, Dr. James Hughes, and Ethan Jackson. I must especially thank James for the opportunities he has given me as a teaching assistant, and especially for the random conversations we would have. Some day guys I am sure we will actually get actual time to sit down and play games. Coming to Western was the first time I ever travelled far away from anywhere resembling home for me, so I was happy for the time we could spend together when we could.

In general, I must thank all the students I have ever taught, past or present; thank you for giving me the opportunity to teach and inspire you. In addition, I want to thank all those that helped me over the years, such as my colleagues in the Department of Computer Science at the University of Manitoba. Thank you Dr. Ben Li for first introducing me to makespan minimization on unrelated parallel machines in 2012.

I need to thank my friends, even if they are far and wide but very few. In that I want to acknowledge my friends online, the “Super Friends”; in particular, I want to thank Tor Asbjørn, Marcus Duplechan, and Jasper Valentine for all the laughs and support these years.

I would like to thank my family. Since being away from Manitoba, my family has grown with new nieces and nephews. Thank you Mom for being there for me and being strong all these years, and for supporting me in my goals even if the light at the end of the tunnel was unclear. During my program, my family lost one of our beloved dogs, Java (2006–2016). She (Java) was my “research buddy” and one of the last fragments reminding me of my Father, Michael William Page (1956–2010). My Father’s pragmatism was one of the things that inspired me to pursue research in approximation algorithms, the algorithms “equivalent” of the things he would build in his spare time to solve tangible problems for others.

Finally I must thank Charén Rinckens. She made plenty of sacrifices by coming with me to London, Ontario since I began my journey home in Winnipeg in 2012. Our move to London in late 2014 was our first big journey together on our own. Thank you for all the hours of me asking you to patiently listening to me explain my proofs and ideas to help convince myself, and for offering an extra pair of eyes when I might have needed them sometimes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Parallel machine scheduling is an integral part of both industrial production and technology, especially as automated scheduling and parallel computing continue to become more relevant. Despite this, in many cases little is known mathematically as to why many hard parallel machine scheduling problems are indeed as difficult as they seem to be, and there is a need for efficient algorithms that have rigourous performance guarantees in many situations. In this thesis we focus on a classic problem in scheduling theory called makespan minimization on unrelated parallel machines and several of its variants, and tread the boundaries between computationally tractable and intractable problems from both an exact and approximate algorithmic standpoint.

In Chapter 1.1 we provide background on optimization problems, computational complexity theory, approximation algorithms, parallel machine scheduling problems, and present a framework that many of the algorithms in this thesis use called a ρ-relaxed decision procedure. Next, in Chapter 1.2 we provide the motivation of the research presented in this thesis, and describe the relevant core scheduling problems that serve as the basis of our work. Finally, in Chapter 1.3 we summarize the problems we investigated, our results, and give an overview of this thesis.

1.1 Background

1.1.1 Optimization Problems and Some Complexity Theory

Parallel machine scheduling problems are optimization problems. An optimization problem [67] has three parts:

- A non-empty set I of *instances* to be considered for the problem. For each instance $x \in I$, let S_x be the set of *feasible solutions* of x.
- A computable function $v : \{(x, y) \mid x \in I, y \in S_x\} \to \mathbb{Q}$ called an *objective function*.
- A *goal* $\in \{\min, \max\}$ that is to either minimize (min) or maximize (max) the objective function.

Let $OPT(x) := \text{goal}\{v(x, y) \mid y \in S_x\}$ be the *optimal objective value* of instance $x \in I$; as shorthand, we often write OPT for the optimal objective value. The goal is to find a feasible solution $y' \in S_x$ with $v(x, y') = OPT(x)$ called an *optimal solution*. As an example, $R||C_{max}$ is a minimization problem where the goal is to find a feasible schedule whose makespan (objective value) is of minimum size. Note that the *decision variant of an optimization problem* is a decision problem that given a value β asks if there is a feasible solution with objective value at most β (for a minimization problem) or at least β (for a maximization problem).

For an instance $x \in I$, let $|x|$ be its *input size*, i.e., the number of bits required to represent x in some fixed encoding. NP is the set of decision problems that can be solved in non-deterministic polynomial time with respect to the input size. All optimization problems with decision variants in NP are called NP-optimization problems (NPO); every problem we discuss in this thesis is NPO. An optimization problem is NP-hard if it is at least as hard as the most difficult problems in NP. That is, an optimization problem Π_1 is NP-hard if all the decision problems Π_2 in NP polynomially reduce to it, i.e. for every $\Pi_2 \in$ NP, there is a polynomial-time algorithm that can take any instance x_2 of Π_2 and transforms it into into an instance of Π_1; thus, if there is a polynomial-time algorithm for Π_1, then there is a polynomial-time algorithm for Π_2. Many optimization problems of both practical and theoretic importance, including $R||C_{max}$, are NP-hard problems.

The decision variant of a NP-hard optimization problem is NP-complete if it is in NP. The problems we study in this thesis have parameters that involve numbers e.g. job lengths, so there are two other kinds of NP-hardness to discuss. An optimization problem is *strongly* NP-*hard* if it remains NP-hard for instances $I' \subseteq I$ where, for each $x' \in I'$, the sizes of the numbers in x' are bounded by a polynomial in $|x'|$. One can similarly define NP-complete problems in the strong sense. All NP-optimization problems that are NP-hard but not strongly NP-hard are called *weakly* NP-*hard*; this is sometimes referred to as NP-hard in the *ordinary sense*.

NP-complete problems are not polynomial-time solvable unless P = NP [67, Section 15.7]. It is widely believed that P ≠ NP [30], but the P vs. NP problem still remains open. Under the assumption that P ≠ NP, researchers design approximation algorithms that efficiently find approximate solutions for NP-hard optimization problems.

1.1.2 Approximation Algorithms

As all the problems we explore are minimization problems, we limit our discussion here strictly to such problems. An *approximation algorithm* is a finite step-by-step procedure that returns for all instances to a given problem an approximate solution in polynomial time. More formally, let $ALG(x)$ be the objective value of a feasible solution produced by the algorithm for instance $x \in I$, and let $OPT(x)$ be the optimum value of instance x. For any instance $x \in I$ of a minimization problem, a *ρ-approximation algorithm* produces a feasible solution in polynomial time such that $1 \leq ALG(x)/OPT(x) \leq \rho$, where ρ is called its *approximation ratio*. For instance, a 2-approximation algorithm will return a feasible solution y with value $v(x, y)$, where $OPT \leq v(x, y) \leq 2 \cdot OPT(x)$ i.e., the value of a feasible solution is always within twice the optimum. The goal is to find ρ-approximation algorithms where ρ is as small as possible.

A *polynomial-time approximation scheme* (PTAS) is a family $\{A_\epsilon\}$ of approximation algorithms, such that algorithm A_ϵ is a $(1 + \epsilon)$-approximation algorithm for every fixed $\epsilon > 0$. Note that there are two special classes of PTASs: a fully polynomial-time approximation scheme

(FPTAS) has polynomial running time also in $1/\epsilon$; and an efficient polynomial-time approximation scheme (EPTAS) has time complexity of the form $f(1/\epsilon) \cdot poly(|x|)$, where $f(1/\epsilon)$ is some computable function that is not necessarily polynomial. Note that all FPTASs are EPTASs, and all EPTASs are PTASs. For example, a PTAS can have running time $O(n^{1/\epsilon})$, but this would not be a FPTAS because its running time is exponential in $1/\epsilon$. Finally, it is worth noting that FPTAS do not exist for strongly NP-hard problems, unless P = NP [28].

1.1.3 Parallel Machine Scheduling Problems

We consider the problem of scheduling parallel machines. When jobs are scheduled on parallel machines, the jobs are processed in parallel and the machines operate independent of one another. We are interested in the case when each job is scheduled *non-preemptively* on a machine, i.e. if a job is scheduled on a machine, then the job must complete without being interrupted.

Let there be m machines $M = \{M_1, \ldots, M_m\}$ and n jobs $J = \{J_1, J_2, \ldots, J_n\}$, where each job J_j is indexed $j \in \{1, 2, \ldots, n\}$ and machine M_i is indexed $i \in \{1, 2, \ldots, m\}$. Parallel machines are called *unrelated* when given a job J_j and machine M_i, the *processing time* for job J_j when scheduled on machine M_i is $p_{i,j} \in \mathbb{Z}^+$, and assume $p_{i,j} = \infty$ implies J_j cannot be scheduled on M_i. One way of representing the processing times of the jobs on the machines is using a $m \times n$ matrix $P = (p_{i,j})$ called a *processing matrix*. For example, let there be $m = 4$ machines and $n = 7$ jobs with the processing matrix

$$P = \begin{array}{c} \begin{matrix} J_1 & J_2 & J_3 & J_4 & J_5 & J_6 & J_7 \end{matrix} \\ \begin{pmatrix} 1 & 1 & 2 & 5 & 2 & 5 & 1 \\ 2 & 3 & 2 & 6 & 1 & 1 & 2 \\ 8 & 1 & 2 & 2 & 2 & 5 & 2 \\ 3 & 2 & 2 & 3 & 3 & 5 & 2 \end{pmatrix} \end{array} \begin{matrix} M_1 \\ M_2 \\ M_3 \\ M_4 \end{matrix}$$

Job J_4, for example, takes $p_{2,4} = 6$ time units on machine M_2, but $p_{3,4} = 2$ time units on machine M_3. We note that there are also two other widely studied parallel machine environments. Parallel machines are called *identical* when every job J_j has a *length* $p_j \in \mathbb{Z}^+$, where its processing time is p_j on any machine. When the parallel machines are *uniform*, every machine has a *speed* $s_i \in \mathbb{Z}^+$, and if J_j is scheduled on machine M_i, its processing time is p_j/s_i. Clearly, the identical parallel machine environment is a special case of the uniform parallel machine environment, and the uniform parallel machine environment is a special case of the unrelated parallel machine environment.

A *schedule* assigns each job to a machine. Given a schedule, the completion time C_j of a job J_j that finishes last is denoted as C_{max} and it is called the *length of the schedule* or *makespan*. While there are different metrics for measuring the quality of a schedule, the makespan is a core metric because typically we desire schedules that complete the jobs as early as possible. More formally, the goal is to produce a schedule $S : J \to M$ such that the makespan, $\max_{1 \le i \le m} \sum_{J_j | S(J_j) = M_i} p_{i,j}$, is minimized. In Figure 1.1 we provide two examples of schedules, the latter being a schedule with minimum makespan—an *optimal schedule*.

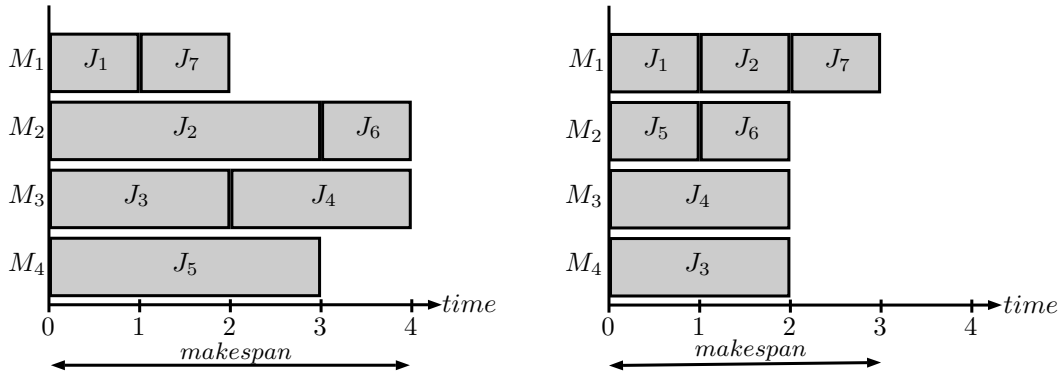

Figure 1.1: Gantt charts showing two schedules for the instance given above. The makespan of the schedules are 4 and 3, respectively. The schedule on the right is an optimal schedule.

We adopt the scheduling-theoretic notation of Graham *et al.* [39]. Graham's notation describes scheduling problems by three fields, $\alpha|\beta|\gamma$, where α specifies a machine environment, β are the job characteristics, and γ is the optimality criterion. We summarize below in more detail the first two of these fields. The third field, the *optimality criterion*, specifies the objective function or goal for a scheduling problem; in this thesis we look at the makespan (C_{max}), so $\gamma = C_{max}$. Below let $\circ$ denote the empty symbol.

Machine environment. A machine environment $\alpha = \alpha_1\alpha_2$ gives the type of machine $\alpha_1 \in \{\circ, P, Q, R\}$, each defined in Table 1.1, and α_2 is either $\circ$ or a positive integer. That is, if $\alpha_2 = \circ$ then the number of machines is variable; otherwise, either α_2 is a positive integer or m:

- If α_2 is a positive integer, then there is a constant number α_2 of parallel machines.
- If $\alpha_2 = m$ then the number of machines can be any number but is fixed, e.g. $Rm||C_{max}$ is makespan minimization on unrelated parallel machines where the number of parallel machines is constant, but $R||C_{max}$ assumes m is not a constant.

α_1	**Machine Environment**	**Definition**
R	Unrelated Parallel Machines	Processing times are given as processing matrix $P = (p_{i,j})$. If job J_j can be scheduled on M_i, it takes $p_{i,j} \in \mathbb{Z}^+$ time units.
Q	Uniform Parallel Machines	Every job J_j has a length $p_j \in \mathbb{Z}^+$, and every machine M_i has a speed $s_i \in \mathbb{Z}^+$. If J_j can be scheduled on M_i, its processing time is p_j/s_i.
P	Identical Parallel Machines	All machines have the same speed, so that $s_1 = s_2 = \cdots = s_m$.
$\circ$	Single Machine	There is $m = 1$ machine and $\alpha_2 = 1$.

Table 1.1: Some machine environments in Graham's notation. Machine environments are given from most general to least.

	Job Characteristic	**Definition**
bag	Bag Constraints	Jobs J are partitioned into ℓ bags $B = (B_1, B_2, \ldots, B_\ell)$, and no two jobs from the same bag can be scheduled on the same machine.
$\mathcal{M}_j$	Eligibility Constraints	Every job $J_j \in J$ has a subset $\mathcal{M}_j \subseteq M$ of machines that J_j can be scheduled on.
pmtn	Preemptive Jobs	Jobs can be interrupted, and can be processed on either the same machine or another machine. If not provided, it is assumed the jobs are to be scheduled non-preemptively.
prec	Precedence Constraints	A partial-ordering is placed on the jobs. Given a directed acyclic graph (DAG) G, an arc $(J_j, J_{j'})$ in G implies J_j must finish before $J_{j'}$ starts.
$p_{i,j} \in A$	Restricted Processing Times	Instances have processing times that are in set A. Similar to this, if $p_j \in A$ is given then it means the job lengths can be values in A.
$p_j = 1$	Unit-Length Jobs	Every job $J_j \in J$ has length $p_j = 1$. All the jobs have the same length.

Table 1.2: Job characteristics.

Job characteristics. The job characteristics specify constraints on the scheduling problem with respect to the jobs, these may generalize the problem or may even restrict the processing times of the jobs on machines[1]. We summarize the ones we use in Table 1.2.

We study in this thesis the problem of scheduling non-preemptively jobs on unrelated parallel machines so as to minimize the makespan. This problem is called *makespan minimization on unrelated parallel machines*, and is abbreviated as $R||C_{max}$. $R||C_{max}$ is a NP-hard problem [29] that has many applications such as in computer scheduling and scheduling production systems with the goal of maximizing productivity.

1.1.4 A ρ-Relaxed Decision Procedure

Many times the algorithms presented in this thesis use algorithms called *ρ-relaxed decision procedures*, these have been used in approximation algorithms for $R||C_{max}$ previously [72]. Let $U \in \mathbb{Z}^+$ be an upper bound on the optimum makespan. By default, we assume $U = p_{max}n$ where $p_{max} = \max_{p_{i,j}|p_{i,j}\neq\infty} p_{i,j}$ and n is the number of jobs[2]. We use binary search over the interval $[0, U]$ to determine the smallest value $\tau \in \mathbb{Z}^+$ for which a ρ-relaxed decision algorithm produces a schedule. A ρ-relaxed decision algorithm either:

[1] In [39] job characteristics are grouped into different categories. For brevity, we will ignore this.

[2] Note that U can also be computed for $R||C_{max}$ by a straight-forward greedy schedule (e.g. see the algorithm of Ibarra and Kim in Chapter 2.2.1), or in machine environments where the machines are identical it is also sufficient to use the sum of job lengths.

- computes a schedule with makespan at most $\rho\tau$; or
- returns `FAIL` if there is no solution with value at most τ.

In the binary search, if the ρ-relaxed decision algorithm returns `FAIL` then the value τ is increased, and if a schedule is returned the value τ is decreased. If we keep track of the schedule with minimum makespan found, after $O(\log U)$ iterations the binary search guarantees that $\tau \leq OPT$ and a schedule with makespan at most $\tau \cdot OPT$ is found. Therefore, if the overall running time of the ρ-relaxed decision algorithm takes polynomial time, this is a ρ-approximation algorithm. Thus, it is sufficient to present a ρ-relaxed decision algorithm for approximation algorithms using this framework.

1.2 Motivation

We focus our research on $R||C_{max}$ to the design of approximation algorithms and the study of inapproximability. Unlike general heuristics, approximation algorithms guarantee that, for any instance of a given problem, the algorithm computes in polynomial time a feasible solution with a guaranteed quality with respect to the optimum called its approximation ratio. We are fundamentally interested in the existence of polynomial-time algorithms with such guarantees. This thesis is not a general expository of heuristic algorithms, for some discussion on heuristic approaches for $R||C_{max}$ we recommend the introduction section of Fanjul-Peyro and Ruiz [26] for a brief but comprehensive summary of heuristics for $R||C_{max}$[3].

Currently the best approximation algorithms for $R||C_{max}$ have approximation ratio 2, and it is known that there is no approximation algorithm with approximation ratio less than $3/2$ unless $\mathsf{P} = \mathsf{NP}$ [72]. Thus a $3/2$-to-2 hardness gap currently exists for $R||C_{max}$. Reconciling or narrowing this gap has remained an open problem for almost thirty years and is regarded as one of the most challenging problems in approximation algorithms today (e.g. Open Problem 10 in Williamson and Shmoys [100], Open Problem 4 in Schuurman and Woeginger [91]).

Problem *Design an approximation algorithm with constant approximation ratio less than* 2 *for* $R||C_{max}$ *or prove that there is no approximation algorithm with approximation ratio smaller than* $\rho > 3/2$*, for any* $\rho > 3/2$.

In this thesis, we are interested firstmost in shedding light on this famous problem. We study special cases of $R||C_{max}$ which, only in recent years, have become central in investigations into the $3/2$-to-2 hardness gap. Two of these problems are known as the restricted assignment problem and the graph balancing problem:

- *Restricted Assignment Problem* ($P|\mathscr{M}_j|C_{max}$). A special case of $R||C_{max}$ where the processing matrix has every $p_{i,j} \in \{p_j, \infty\}$, where each $p_j \in \mathbb{Z}^+$ is the length of $J_j \in J$. This problem can be viewed as makespan minimization on identical parallel machines with the constraint that, for each job $J_j \in J$, there is a set $\mathscr{M}_j$ of machines where J_j can be scheduled; we call $\mathscr{M}_j$ the eligibility constraints or processing sets of J_j. Assigning each job J_j to an eligible machine in $\mathscr{M}_j$ is said to satisfy the *eligibility constraints*.

[3] According to Fanjul-Peyro and Ruiz, some of the most successful types of heuristic algorithms for $R||C_{max}$ include cutting-plane algorithms [80], local search algorithms [25], and recovering beam search algorithms [32].

- *Graph Balancing Problem* ($P|\mathscr{M}_j, |\mathscr{M}_j| \leq 2|C_{max}$): This is a special case of the restricted assignment problem where the number of eligible machines for each job is at most 2. An alternate way to interpret an instance of this problem is as a weighted multigraph where the jobs are edges and the machines are vertices, and every edge must be directed to one of its endpoints so as to minimize the maximum sum of the lengths of the edges directed toward a vertex. More formally, let weighted multigraph $G = (V, E, \mathbf{p}, \mathbf{q})$ where $\mathbf{p} = (p_{e_1}, \ldots, p_{e_{|E|}})$ are the *lengths* for the edges with $p_{e_j} \in \mathbb{Z}^+$ and $\mathbf{q} = (q_{v_1}, \ldots, q_{v_{|V|}})$ are the *dedicated loads*[4] where $q_{v_i} \in \mathbb{Z}^+ \cup \{0\}$. The goal is to find an *orientation* $\sigma : E \to V$ such that the maximal load (makespan) is minimized[5], where the *load of a vertex* $v \in V$ is $q_v + \sum_{e|\sigma(e)=v} p_e$.

Note that if we refer to the above two problems "on uniform parallel machines" or "with speeds", this simply means that the machines have speeds and a job J_j takes p_j/s_i instead of p_j on machine M_i. Also, if we refer to problems "with two job lengths", this means that every job length $p_j \in \{\ell_s, \ell_b\}$, where $\ell_s < \ell_b$.

1.3 Problems Investigated, Our Results, and Overview

The best-known approximation algorithm for the graph balancing problem is the algorithm of Ebenlendr *et al.* [21] which has approximation ratio 7/4. It has also been established that there is no ρ-approximation algorithm with $\rho < 3/2$ for the graph balancing problem with two job lengths either 1 or 2, unless P = NP [2, 21]. The graph balancing problem is one of the first-known special cases of $R||C_{max}$ that shares the same 3/2-inapproximability (3/2-hardness) bound and for which there is an approximation algorithm with approximation ratio strictly less than 2. Since then, there have been no improvements upon the approximation ratio for this problem, but there still was an outstanding 3/2-to-7/4 hardness gap for the graph balancing problem with two job lengths. In 2013, Kolliopoulos and Moysoglou [66] presented a 1.652-approximation algorithm for the graph balancing problem with two job lengths. Our first goal was to improve upon their approximation ratio and successfully close the hardness gap for the graph balancing problem with two job lengths. In Chapter 3 we present two 3/2-approximation algorithms for the graph balancing problem with two job lengths.

Naturally, upon successfully closing the hardness gap for the graph balancing problem with two job lengths, we pursued other cases of the graph balancing problem. One aspect of the 3/2-to-2 hardness gap of $R||C_{max}$ that we know very little about is if there exist approximation algorithms with approximation ratios less than 2 when the machines are uniform. Two special cases of $R||C_{max}$ we consider in Chapter 4 are the following:

- *Restricted assignment problem with two job lengths on uniform parallel machines* ($Q|\mathscr{M}_j, p_j \in \{\ell_s, \ell_b\}|C_{max}$): a job J_j takes $p_{i,j} = p_j/s_i$ time units on machine $M_i \in \mathscr{M}_j$ where $s_i \in \mathbb{Z}^+$ is the speed of M_i and $p_j \in \{\ell_s, \ell_b\}$ with $\ell_s < \ell_b$, and $p_{i,j} = \infty$ otherwise.

[4]Dedicated loads allow us to assume there are no self-loops in the multigraph. Sometimes, when explicitly stated, we will assume there are self-loops present in the multigraph, implying that $q_v = 0$, $v \in V$.

[5]Technically indexing edges and vertices by an edge or vertex directly is an abuse of notation, i.e. writing "edge e with length p_e" or "vertex v with dedicated load q_v", as opposed to writing "edge E_e with length p_e" or "vertex V_v with dedicated load v" and treating e and v as positive integers.

- *Graph balancing problem with two speeds and two weights*: every vertex $v \in V$ has a speed s_v, where s_v is one of two possible speeds $\mathfrak{z}_s, \mathfrak{z}_f \in \mathbb{Z}^+$ with $\mathfrak{z}_s < \mathfrak{z}_f$, and if edge e with length $p_e \in \{\ell_s, \ell_b\}$ is oriented towards v, it contributes to the load p_e/s_v. Note that the dedicated load of v contributes q_v/s_v to the load of v, and $\ell_s < \ell_b$. We denote the weighted multigraph G for this problem as $G = (V, E, \mathbf{p}, \mathbf{q}, \mathbf{s})$, where $\mathbf{s} = (s_{v_1}, \ldots, s_{v_{|V|}})$ are the speeds of the vertices.

We present a $(2 - \ell_s/\ell_b)$-approximation algorithm for the restricted assignment problem with two job lengths on uniform parallel machines, then we give a $(\sqrt{65} + 7)/8$-approximation algorithm for the graph balancing problem with two speeds and two job lengths.

We are also interested in identifying cases with very specialized structure where we can still show that the 3/2-inapproximability of $R||C_{max}$ persists but there are 3/2-approximation algorithms. Motivated by the work of Jansen *et al.* [53], we consider a job-intersection-graph-based model with $R||C_{max}$. In Chapter 5 we study the following two problems:

- $R||C_{max}$ *with simple job-intersection structure.* A *job-intersection graph* $G_J = (J, E_J)$ has a vertex for each job $J_j \in J$, and for any two jobs $J_j, J_{j'} \in J$, there is an edge $\{J_j, J_{j'}\} \in E_J$ if there is a machine M_i such that $p_{i,j} \neq \infty$ and $p_{i,j'} \neq \infty$. A set of restrictions on which machines can process a job can be represented as a job-intersection graph. We study $R||C_{max}$ *restricted to* particular classes of job-intersection graphs. We give an example of a job-intersection graph in Figure 1.2.

Instance: $m = 3, n = 4$

$$P = \begin{array}{c} \begin{matrix} J_1 & J_2 & J_3 & J_4 \end{matrix} \\ \begin{pmatrix} 1 & 2 & 1 & \infty \\ 3 & 5 & 1 & \infty \\ \infty & \infty & 1 & 4 \end{pmatrix} \end{array} \begin{matrix} \\ M_1 \\ M_2 \\ M_3 \end{matrix}$$

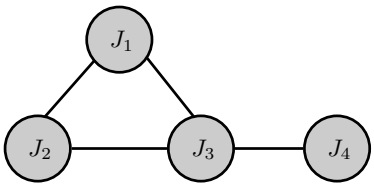

Figure 1.2: An instance of $R||C_{max}$ (left), and its job-intersection graph G_J (right).

- $R||C_{max}$ *with bounded job assignments.* Let $\mathcal{J}_i$ be the set of jobs that can be processed by machine M_i, i.e., $\mathcal{J}_i = \{j \in J \mid p_{i,j} \neq \infty\}$. Let $\kappa > 0$. We consider $R||C_{max}$ restricted to instances when, for each machine M_i, $|\mathcal{J}_i| \leq \kappa$. Clearly when $\kappa = n$, it is $R||C_{max}$.

A graph is *triangle free* if it does not contain any simple cycles of length 3—triangles. Note that all bipartite graphs contain no odd-length cycles, thus all bipartite graphs are triangle free. The *diamond graph* consists of four vertices and five edges, so it is K_4 less one edge. We call a graph *diamondless* if it does not contain the diamond graph as a subgraph. In contrast, a *diamond-free graph* is defined as not having the diamond graph as an induced subgraph. An *induced subgraph* $H = (V', E')$ of a graph $G = (V, E)$ is such that $V' \subseteq V$ and an edge $e = \{u, v\} \in E'$ if both $u, v \in V'$ and $e \in E$; all diamondless graphs are diamond-free, but not all diamond-free graphs are diamondless. For example, the graph K_4 is diamond free but is not diamondless. In Figure 1.3 we give an instance of the graph balancing problem where its job-intersection graph is both diamondless and diamond free.

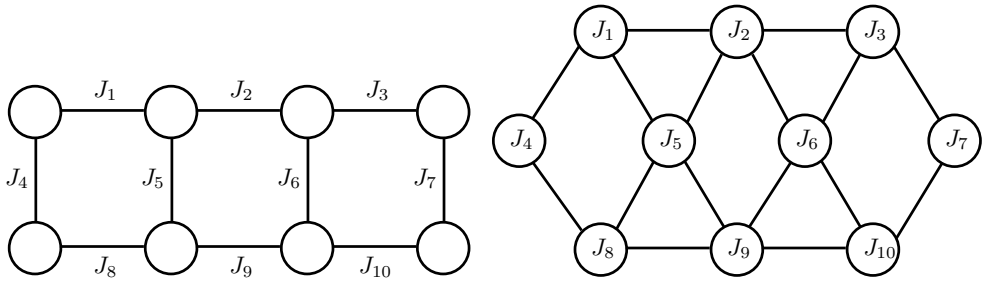

Figure 1.3: An instance of the graph balancing problem (left) and its job-intersection graph (right).

In Chapter 5 we identify that the 3/2-inapproximability of $R||C_{max}$ persists when we allow a few eligible jobs per machine. That is, even for the graph balancing problem with $r = 3$ and the graph balancing problem restricted to diamondless job-intersection graphs, there is no ρ-approximation algorithm with $\rho < 3/2$ unless P = NP; we obtain a 3/2-approximation algorithm in both cases for $R||C_{max}$, and this approximation algorithm has approximation ratios 5/3 and 1 for $R||C_{max}$ with $r = 4$ and $r = 2$, respectively. It is worth noting that the same algorithm is also a polynomial-time algorithm for $R||C_{max}$ restricted to triangle-free job-intersection graphs. Among these results, we give a $(2 - 1/(r - 1))$-approximation algorithm for the restricted assignment problem with two job lengths when $r \geq 3$, and prove that $R||C_{max}$ restricted to the following job-intersection graph classes is 3/2-inapproximable: complete graphs, threshold graphs, interval graphs, cographs, split graphs, and house-free graphs.

Finally, we consider a generalization of $R||C_{max}$ where the jobs are partitioned into b sets $B = (B_1, B_2, \ldots, B_b)$ called *bags*, and any feasible solution must satisfy the *bag constraints*: no two jobs from the same bag can be scheduled on the same machine. This problem is called *makespan minimization on unrelated parallel machines with bags*, and we denote it as $R|bag|C_{max}$. Notice that if $b = |J|$, then every job is in a distinct bag, and we get $R||C_{max}$. As discussed in [22], the bag constraints appear in settings such as in the scheduling of tasks for on-board computers in airplanes. In these systems it is required for some tasks to be scheduled on different processors so that the airplane continues to operate safely even if one of the processors were to fail. Thus, parallel machine scheduling problems with bag constraints model fault-tolerant scheduling needed in complex parallel systems where system stability is desired [17].

We study $R||C_{max}$, the restricted assignment problem, and the graph balancing problem when bags are introduced, with an emphasis on when the number b of bags is small. In addition to this, we investigate $R|bag|C_{max}$ in the setting with so-called machine types. As discussed by Gehrke *et al.* [31], a natural scenario in parallel machine scheduling is where the machines are clusters of processors where each processor in a cluster is of the same type, e.g. clusters of CPUs and/or GPUs. More formally, two machines M_i and $M_{i'}$ have the same *machine type* if $p_{i,j} = p_{i',j}$ for all $J_j \in J$. We study $R|bag|C_{max}$ *with machine types*, where the number δ of machine types is constant.

In Chapter 6 we present a variety of algorithmic results, and we illustrate some complexities of the problem by bridging the inapproximability results known for the problems such as the graph balancing problem and the restricted assignment problem to these scheduling problems using only a few bags. We present a b-approximation algorithm for $R|bag|C_{max}$, a PTAS for

$R|bag|C_{max}$ when both δ and ℓ are constant, and a $\ell/2$-approximation algorithm for the graph balancing problem with $\ell \geq 2$ bags. In addition, we present polynomial-time algorithms for the restricted assignment problem on uniform parallel machines with bags when all the jobs have unit length ($Q|bag, \mathcal{M}_j, p_j = 1|C_{max}$), and $P|bag|C_{max}$ with $\ell = 2$ bags. To complement our results, we prove that there is no ρ-approximation algorithm with $\rho < 3/2$ for the restricted assignment and graph balancing problems with $\ell = 2$ and $\ell = 3$ bags, unless $\mathsf{P} = \mathsf{NP}$, and that both $P|bag|C_{max}$ with $\ell = 3$ bags and $Q|bag|C_{max}$ with $\ell = 2$ bags are strongly NP-hard.

While we ultimately do not resolve the 3/2-to-2 hardness gap of $R||C_{max}$, we present some examples of special cases of $R||C_{max}$ where 3/2-approximation algorithms do exist, and some for broader classes of instances where there are approximation algorithms with approximation ratios less than 2; many of our algorithms use the ρ-relaxed decision procedure given in Chapter 1.1.4. We also illustrate through our study of $R||C_{max}$ with simple job-intersection structure how, even in instances with very specific job-intersection structures, the 3/2-inapproximability for $R||C_{max}$ persists. In addition, we demonstrate the polynomial-time solvability of a variety of scheduling problems, and present results for $R|bag|C_{max}$.

We list our approximation algorithm results in Table 1.3, polynomial-time solvability results in Table 1.4, and our computational complexity results in Table 1.5. When in the tables below we state an algorithm is *combinatorial*, we mean that it does not use an algorithm to solve a linear program nor does it employ methods from linear algebra or convex geometry; algorithms of this kind might compute flows or matchings, and comparisons and basic arithmetic operations are done over rational numbers. We make this distinction at times as algorithms in this domain employ linear programming frequently, and combinatorial algorithms tend to be simpler and easier to implement.

Most of the work presented in this thesis has been previously published. The work presented in Chapter 3.1 was first published in the journal *Algorithms* [84], and the work in Chapter 3.2 has been unpublished until now. The results presented in Chapter 4 were accepted to the *Journal of Combinatorial Optimization* [85]. The results given in Chapter 5 were presented at the *Twelfth International Conference on Combinatorial Optimization and Applications* (COCOA 2018) [87], and the results in Chapter 6 were presented at the *Twelfth International Conference on Algorithmic Aspects in Information and Management* (AAIM 2018) [86].

In Chapter 2 we present a literature review. Then, in Chapters 3–6 we present our results. Finally, we summarize our results and present possible future research directions in Chapter 7.

Ch.	Problem	Prev. Best	New	Notes
3	Graph balancing problem with two job lengths	1.652 [66]	$3/2$ ✓	Two $3/2$-approximation algorithms are presented, the algorithm in Chapter 3.2 is purely combinatorial.
4	Restricted assignment problem with two job lengths on uniform parallel machines	2	$2 - \frac{\ell_s}{\ell_b}$	Combinatorial. Algorithm matches bound $3/2$ when $\ell_s = 1$ and $\ell_b = 2$. ✓
4	Graph balancing problem with two speeds and two job lengths	2	$\frac{\sqrt{65}+7}{8}$	$\frac{\sqrt{65}+7}{8} \approx 1.88278$.
5	$R\|\|C_{max}$ with bounded job assignments ($\kappa = 4$)	2	$5/3$	Combinatorial.
5	$R\|\|C_{max}$ with bounded job assignments ($\kappa = 3$)	2	$3/2$ ✓	Same as algorithm above. This is a $3/2$-approximation algorithm for $R\|\|C_{max}$ restricted to diamond-less job-intersection graphs. ✓
5	Restricted assignment problem with two job lengths and $\kappa \geq 3$	2	$2 - \frac{1}{\kappa - 1}$	Matches bound $3/2$ when $\kappa = 3$. ✓
6	$R\|bag\|C_{max}$	[17] claims $O\left(\frac{\log n}{\log \log n}\right)$, uses randomized rounding and is not yet published.	$\mathcal{O}$	Combinatorial.
6	$R\|bag\|C_{max}$ with fixed δ and $\mathcal{O}$	-	PTAS ✓	Implies there is a PTAS for $Q\|bag\|C_{max}$ with fixed numbers of machine speeds and $\mathcal{O}$ (Corollary 6.3.2). ✓
6	Graph balancing problem with bags	-	$\mathcal{O}/2$	For $\mathcal{O} \geq 2$ bags. Combinatorial, and matches bound $3/2$ for $\mathcal{O} = 3$. ✓

Table 1.3: Summary of approximation algorithm results in this thesis in the order in which they are presented. Results indicated with "✓" match the best-known inapproximability bounds previously established or proven in this thesis.

Ch.	Problem	Notes
5	$R\|\|C_{max}$ with bounded job assignments ($\gamma = 2$)	Combinatorial. This is also a polynomial-time algorithm for $R\|\|C_{max}$ restricted to triangle-free job-intersection graphs (includes bipartite job-intersection graphs).
6	Graph balancing problem with $\mathscr{b} \leq 2$	Combinatorial. Takes $O(m^2n+n^2m)$ time. Implied by $\mathscr{b}/2$-approximation algorithm, and there is a straightforward algorithm for $R\|bag\|C_{max}$ with $\mathscr{b} = 1$ bags (Chapter 6.1).
6	$Q\|bag, \mathscr{M}_j, p_j = 1\|C_{max}$	Combinatorial.
6	$P\|bag\|C_{max}$ with $\mathscr{b} = 2$	Takes $O(m \log m)$ time.

Table 1.4: Summary of exact polynomial-time algorithms presented in this thesis. Results are listed in the order in which they are presented.

Ch.	Problem	Result	Notes
5	Graph balancing problem restricted to diamondless job-intersection graphs with job lengths either 1 or 2	3/2-hardness ✓	There is no ρ-approximation algorithm with $\rho < 3/2$, unless P = NP. Implies 3/2-hardness for $R\|\|C_{max}$ with $\gamma = 3$.
6	Restricted assignment problem with $\mathscr{b} = 2$ bags with two job lengths either 1 or 2	3/2-hardness	There is no ρ-approximation algorithm with $\rho < 3/2$, unless P = NP. Prior to our results, it was known that there is no constant-factor approximation algorithm for arbitrary numbers of bags and job lengths [17]. When there is only one job length, it is polynomial-time solvable on uniform parallel machines (Theorem 6.5.2).
6	Graph balancing problem with $\mathscr{b} = 3$ bags with two job lengths either 1 or 2	3/2-hardness ✓	There is no ρ-approximation algorithm with $\rho < 3/2$, unless P = NP.
6	$P\|bag\|C_{max}$ with $\mathscr{b} = 3$ bags	strongly NP-hard ✓	Independently proven by Dokka *et al.* [19]. Polynomial-time solvable for $\mathscr{b} = 2$ (Theorem 6.5.3). Previously a PTAS for $P\|bag\|C_{max}$ was known [17].
6	$Q\|bag\|C_{max}$ with $\mathscr{b} = 2$ bags	strongly NP-hard	We also show $Q\|bag\|C_{max}$ with $\mathscr{b} = 2$ bags is strongly NP-hard when the job lengths $p_j \in \{1, 2, \ldots, m\}$ (Corollary 6.6.7).

Table 1.5: Summary of complexity results presented in this thesis. For inapproximability results, we indicate "✓" when the inapproximability bound is matched by an approximation algorithm for said problem or a generalization of it.

Chapter 2

Literature Review

Makespan minimization problems on parallel machines are some of the most studied problems in all of scheduling and combinatorial optimization. Parallel machine scheduling has numerous applications such as in mass production lines [94]. For example, Yu *et al.* [101] describes one major bottleneck in printed wiring board (PWB) manufacturing: scheduling the drilling operation. In the drilling operation, a group of parallel machines, the drilling machines, each with different processing speeds and operating characteristics, process lots conveyed in as a batch. For each lot, the drilling machines will drill holes as per required by the specifications of the manufacturer. Note that an individual lot may be processed by a subset of the machines, and each lot may be completed at different speeds due to characteristics between the lots and machines e.g. lot size, board layout, drill hole requirements, general machine specifications. Each time the scheduling system is started, a batch of lots is provided a time window τ to be drilled. The goal is for the system to schedule the lots to drilling machines with the smallest possible value for τ.

In this chapter we begin with makespan minimization on identical parallel machines ($P||C_{max}$) and uniform parallel machines ($Q||C_{max}$) in Chapter 2.1, then $R||C_{max}$ and the main special cases we are interested in within Chapter 2.2.1. Finally, in Chapter 2.3, we summarize the literature for other scheduling problems relevant to Chapters 5–6, and for the sake of completeness we include a basic discussion on preemptive scheduling and precedence-constraint scheduling.

2.1 Scheduling Identical and Uniform Parallel Machines ($P||C_{max}$ and $Q||C_{max}$)

It is known that $P2||C_{max}$ is NP-hard, and $P||C_{max}$ (i.e., m is not fixed) is strongly NP-hard [29]. Two of the earliest-known approximation algorithms in the literature are the approximation algorithms presented by Graham *et al.* [37, 38] for $P||C_{max}$.

- Graham's list scheduling algorithm: For $j = 1, 2, 3, \ldots, n$, schedule job J_j on the machine with least load. This algorithm has approximation ratio 2.
- Graham's longest processing time algorithm (LPT): Sort the jobs from longest to shortest, then apply the list scheduling algorithm. Sorting the jobs in this fashion improves the approximation ratio to $4/3$.

Hochbaum and Shmoys [42] introduced the dual approximation approach for $P||C_{max}$ in order to design a PTAS. Recently, Jansen and Rohwedder [60] presented an EPTAS for $P||C_{max}$ with running time bounded by $2^{O((1/\epsilon)\log^2(1/\epsilon))} + O(n)$.

There also is a PTAS for $Q||C_{max}$ due to Hochbaum and Shmoys [43], it has running time $(n/\epsilon)^{O(1/\epsilon^2)}$. In 2010, Jansen [49] presented an EPTAS for $Q||C_{max}$ that has running time bounded by $2^{O(1/\epsilon^2\log^3(1/\epsilon))} + poly(n)$. Presently the fastest-known EPTAS for $Q||C_{max}$ is given by Jansen *et al.* [50], with running time $2^{O((1/\epsilon)\log^4(1/\epsilon))} + poly(n)$.

2.2 Scheduling Unrelated Parallel Machines

Here we discuss results for $R||C_{max}$, and for special cases of $R||C_{max}$ that relate to the restricted assignment problem and the graph balancing problem. For the sake of brevity, here we keep our discussion of the literature narrow. For example, $Rm||C_{max}$ (i.e. $R||C_{max}$ with a fixed number of machines) has FPTASs [55]; thus, this section does not discuss cases in which the number of machines is constant.

2.2.1 Makespan Minimization on Unrelated Parallel Machines ($R||C_{max}$)

Earlier Research

The first approximation algorithms for $R||C_{max}$ were given in 1977 by Ibarra and Kim [48]. Of these, an m-approximation algorithm that does the following is given: For each job $J_j \in J$, schedule J_j on the machine for which J_j completes the earliest. Later, Davis and Jaffe [18] gave a $2\sqrt{m}$-approximation algorithm for $R||C_{max}$. This is accomplished by extending the idea of Ibarra and Kim but adding some preprocessing to estimate potential job assignments and introducing thresholds that activate and deactivate the machines as a schedule is being produced, and obtaining their desired approximation ratio through a fairly technical analysis. The first constant-factor approximation algorithm for $R||C_{max}$ was given by Lenstra *et al.* [72], it has approximation ratio 2. Lenstra *et al.* also proved that there is no ρ-approximation algorithm with $\rho < 3/2$ for the restricted assignment problem with two job lengths $p_j \in \{1, 2\}$ unless $\mathsf{P} = \mathsf{NP}$; their reduction employs the 3-dimensional matching problem, similar to the one we provide in Chapter 6.6.1.

2-Approximation Algorithm of Lenstra, Shmoys, and Tardos

One common but powerful technique used in approximation algorithms for $R||C_{max}$ and special cases of it is linear program rounding (LP-rounding). In LP-rounding, the idea is to solve a linear program (LP) that is the relaxation of an integer program (IP) formulation of the problem, then round the solution of the LP to obtain a feasible solution. The approximation ratio is usually determined using properties of the LP and the problem, and how it is rounded. To illustrate this technique, we briefly describe the 2-approximation algorithm of Lenstra *et al.* [72]; this algorithm uses a 2-relaxed decision procedure.

Before we proceed, we formulate an IP formulation of $R||C_{max}$. Let $\tau \in \mathbb{Z}^+$ be an estimate for the optimum makespan. We associate the IP with this value τ. Also we introduce binary

indicator variables $x_{i,j} \in \{0,1\}$ for $1 \leq i \leq m$ and $1 \leq j \leq n$, where $x_{i,j} = 1$ means job J_j is scheduled on machine M_i, and if $x_{i,j} = 0$, then job J_j is not scheduled on machine M_i.

Integer programming formulation of $R||C_{max}$ (IP1) for a given bound τ on the makespan

$$\sum_{i=1}^{m} x_{i,j} = 1, \qquad \text{for } j = 1, 2, \ldots, n$$

$$\sum_{j=1}^{n} p_{i,j} x_{i,j} \leq \tau, \qquad \text{for } i = 1, 2, \ldots m$$

$$x_{i,j} \in \{0,1\}, \qquad \text{for all } 1 \leq i \leq m,\ 1 \leq j \leq n.$$

The first set of constraints guarantee that a job is scheduled on one of the machines, and the second set of constraints state that the load of a machine cannot exceed the value τ. Since every job must be scheduled on a machine and the makespan does not exceed τ, if one finds the smallest value for τ for which this IP has a feasible solution, then an optimal schedule can be obtained by scheduling each job J_j on machine M_i if $x_{i,j} = 1$. Unfortunately, we know $R||C_{max}$ is NP-hard and the problem of solving any IP is NP-hard [62], so this approach cannot be used to yield a polynomial-time algorithm. Fortunately, a relaxation of an IP called a linear program (LP) can be solved in polynomial time [61, 63]. In this *relaxation* (called a *LP-relaxation*) each binary indicator variable $x_{i,j} \in \{0,1\}$ is replaced with a variable that satisfies $0 \leq x_{i,j} \leq 1$. The algorithm described here will use a specific type of solution to a LP called an extreme point solution.

Observe that a feasible solution of a LP may have variables $x_{i,j} \in \{0,1\}$, but it may also have some variables $x_{i,j} \in [0,1]$. To produce a feasible solution for $R||C_{max}$ the variables with value $0 < x_{i,j} < 1$ need to be rounded to integer values. If one seeks out the smallest value for τ for which the LP-relaxation of IP1 has a solution, one can show that the worst-case ratio between the optimal solutions of IP1 and its LP-relaxation, its so-called integrality gap (defined more carefully in Chapter 2.2.2) is at least m. This can be observed by considering an instance with one job that has processing time m on any of the machines. Observe that an optimal solution of IP1 has makespan $\tau = m$, but the LP-relaxation can find a feasible solution when $\tau = 1$ by assigning each $x_{i,j}$ value $1/m$. Hence, the above LP-relaxation has an unbounded integrality gap and so a stronger LP-formulation is desired.

A problem with the LP-formulation above is that it does not encode the fact that if $p_{i,j} > \tau$, then $x_{i,j} = 0$. To formulate this constraint, we only consider entries of the processing matrix for which $p_{i,j} \leq \tau$. Hence, we only consider the machines and jobs given in pairs of the set $S_\tau = \{(i,j) \mid p_{i,j} \leq \tau\}$. Thus, we can restrict the machines and jobs used to $|S_\tau|$ variables in IP1, and obtain LP1 below.

Assignment Linear Program (LP1)

$$\sum_{i|(i,j)\in S_\tau} x_{i,j} = 1, \qquad \text{for } j = 1, 2, \ldots, n$$

$$\sum_{j|(i,j)\in S_\tau} p_{i,j} x_{i,j} \leq \tau, \qquad \text{for } i = 1, 2, \ldots m$$

$$x_{i,j} \geq 0, \qquad \text{for all } (i, j) \in S_\tau,$$

where

$$S_\tau = \Big\{(i, j) \in \{1, 2, \ldots, m\} \times \{1, 2, \ldots, n\} \mid p_{i,j} \leq \tau\Big\}.$$

LP1 is used by Lenstra *et al.* to design a 2-approximation algorithm for $R||C_{max}$. A feasible solution for LP1 is called an *extreme point solution* if and only if $|S_\tau|$ linearly independent constraints in LP1 are tight (i.e., are satisfied with equality). An extreme point solution of LP1 can be found in polynomial time [40, 63]. There are $n + m + |S_\tau|$ constraints, so an extreme point solution of LP1 sets at least $|S_\tau| - (n + m)$ variables in constraints of the form $x_{i,j} \geq 0$ to zero. Each constraint of this form corresponds to a pair in S_τ. Since there is a variable $x_{i,j}$ for each pair in S_τ, at most $(n + m)$ of the variables are non-zero.

Given a feasible solution $\mathbf{x}$ of LP1, a job is *integrally set* if $x_{i,j} \in \{0, 1\}$, and *fractionally set* when $0 < x_{i,j} < 1$. Let us count the number of integrally set jobs and fractionally set jobs if $\mathbf{x}$ is an extreme point solution; we will denote these as ϕ and β, respectively. By definition, each fractionally set job is assigned to at least two machines. So in $\mathbf{x}$, this corresponds to at least two non-zero variables in $\mathbf{x}$. As we described above, there are at most $(n + m)$ non-zero variables in $\mathbf{x}$; hence, $\phi + 2\beta \leq n + m$. By definition, $\phi + \beta = n$, and therefore $\phi \geq n - m$ and $\beta \leq m$. Thus, if the algorithm finds an extreme point solution, then at least $n - m$ jobs are integrally set, and the variables for at most m jobs still need to be rounded.

Now we can describe the 2-relaxed decision algorithm. For estimate τ, the algorithm computes an extreme point solution $\mathbf{x}$ of LP1. If no solution is found, there is no solution for value τ and return `FAIL`; otherwise proceed as follows. For each integrally set job with $x_{i,j} = 1$, schedule job J_j on machine M_i. Next the algorithm constructs a bipartite graph $H = (M \cup F, E)$, where F is the set of fractionally set jobs of $\mathbf{x}$, M is the set of machines, and $E = \{(M_i, J_j) \mid M_i \in M,\ J_j \in F,\ x_{i,j} > 0\}$. The algorithm computes such a perfect matching $\mathscr{M}$ that matches every vertex in F to a vertex in M; the existence of a perfect matching is proved by Lenstra *et al*. For each $(M_i, J_j) \in \mathscr{M}$, job J_j is scheduled on machine M_i.

We now determine the makespan of the schedule produced by this algorithm. Scheduling the integrally set jobs of $\mathbf{x}$ causes the load of each machine to be at most τ by the constraints $\sum_{j|(i,j)\in S_\tau} p_{i,j} x_{i,j} \leq \tau$. The perfect matching schedules at most one fractionally set job per machine, and $p_{i,j} \leq \tau$ because $(i, j) \in S_\tau$. Hence, we obtain the following result.

Theorem 2.2.1 (Lenstra *et al.* [72]) *Let $p_{max} = \max_{p_{i,j}|p_{i,j}\leq\tau}(p_{i,j})$. For makespan estimate τ, the algorithm of Lenstra et al. for $R||C_{max}$ computes a schedule with makespan at most $\tau + p_{max}$, and returns `FAIL` when there is no schedule with makespan at most τ.*

Therefore, the makespan of the schedule is at most

$$\tau + \max_{(i,j)\in S_\tau}(p_{i,j}) = \tau + \max_{p_{i,j}|p_{i,j}\leq\tau}(p_{i,j}) \leq \tau + \tau = 2\tau.$$

Now we show that the overall algorithm terminates in polynomial time. If p_{max} is the largest valid entry in the processing matrix, a binary search combined with the above 2-relaxed decision algorithm performs $O(\log n + \log p_{max})$ many iterations. Computing an extreme point solution to LP1, constructing the bipartite graph H, and finding a perfect matching [44] can all be done in polynomial time. Thus, the algorithm of Lenstra *et al.* terminates in polynomial time and is a 2-approximation algorithm for $R||C_{max}$.

Further Developments

Since the 2-approximation algorithm of Lenstra *et al.* [72], there has been little improvement to this result. Shmoys and Tardos eliminate [93] the necessity of computing the extreme point solution in the 2-approximation algorithm, the most expensive step in the algorithm. They developed an algorithm that takes any feasible solution of LP1 and using matching techniques produces a schedule of the same length stated in Theorem 2.2.1. By refining the rounding procedure, Shchepin and Vakhania [92] slightly improved the approximation algorithm of Lenstra *et al.* by reducing the approximation ratio to $(2 - 1/m)$. Later Gairing *et al.* [27] gave a more efficient flow-based 2-approximation algorithm for $R||C_{max}$.

2.2.2 Restricted Assignment Problem ($P|\mathscr{M}_j|C_{max}$)

Presently the best-known approximation algorithms for the restricted assignment problem are also those for $R||C_{max}$. Recent developments have led to polynomial-time algorithms that can successfully estimate the value of the optimal makespan for the restricted assignment problem within a multiplicative factor strictly less than 2. These results were made possible through employing a stronger LP than LP1 called the configuration LP, described below.

For the restricted assignment problem with two job lengths $\ell_s, \ell_b \in \mathbb{Z}^+$, $\ell_s < \ell_b$, Kolliopoulos and Moysoglou [66] presented a flow-based $(2 - \ell_s/\ell_b)$-approximation algorithm when ℓ_s divides ℓ_b. Later Chakrabarty *et al.* [10] showed there is a $(2 - \alpha)$-approximation algorithm for some small value $\alpha > 0$ and also gave a combinatorial $(2 - \ell_s/\ell_b)$-approximation algorithm; we generalize in Chapter 4.1 this $(2 - \ell_s/\ell_b)$-approximation algorithm so that it also works for uniform machines.

When every job has the same length, Lin and Li [79] presented an algorithm that solves the restricted assignment problem when the parallel machines are uniform ($Q|\mathscr{M}_j, p_j = 1|C_{max}$) in $O(n^3 \log nc)$ time, where c is the least common multiple of the machine speeds.

Researchers have also considered the restricted assignment problem when the eligibility constraints $\mathscr{M}_j$ are not arbitrary. For example, consider an application described by Ou *et al.* [83] for the so-called inclusive processing set case. Consider a vessel with cargo to load or unload using cranes, where the cranes have the same operating speeds but each can only handle a certain weight for a piece of cargo. Each piece of cargo has a weight, and it can be managed by any crane with weight capacity at least the weight of the cargo. The loading/unloading time taken by a crane to load/unload a piece of cargo is crane-independent as it depends on the size and location of the cargo on a vessel. Every crane is a machine, and each piece of cargo is a job where its eligibility constraints $\mathscr{M}_j$ are the machines that can lift that cargo. Problems such as these are parallel machine scheduling with processing set restrictions; we briefly summarize later some of the main results for these special cases of the restricted assignment problem.

The Configuration Linear Program and Better Estimates of the Optimum Makespan

Another avenue of research into the restricted assignment problem along and other special cases of $R||C_{max}$ has been attempts to utilize a stronger linear program called the configuration LP (given as LP2, below). As we discuss below, there has been partial success in breaking the approximation barrier of 2 for the restricted assignment problem using the configuration LP. In LP1, jobs are assigned to machines so that the loads of the machines do not exceed some makespan τ and this relationship is captured by having variables that represent an assignment of jobs to machines. Instead, in LP2 the idea is to view the machines as each having a configuration of jobs, a set of jobs that cause on a machine a load at most τ, and representing these configurations of jobs as the variables. Then it is just a matter of assigning exactly one configuration to each machine, and ensuring that each job appears in exactly one of these configurations.

Configuration Linear Program (LP2)

$$\sum_{C \in \mathscr{C}_i(\tau)} y_{i,C} = 1, \qquad \text{for all } M_i \in M$$

$$\sum_{M_i \in M} \sum_{C \in \mathscr{C}_i(\tau) | J_j \in C} y_{i,C} = 1, \qquad \text{for all } J_j \in J$$

$$y_{i,C} \geq 0, \qquad \text{for all } M_i \in M, C \in \mathscr{C}_i(\tau),$$

where the set of *configurations* for machine M_i are

$$\mathscr{C}_i(\tau) = \left\{ C \subseteq J \mid \sum_{J_j \in C} p_{i,j} \leq \tau \right\}.$$

The integral version (i.e. replace $y_{i,C} \geq 0$ with $y_{i,C} \in \{0, 1\}$) of LP2 indeed captures $R||C_{max}$: The first set of constraints ensure that one configuration of jobs is assigned to each machine; the second set of constraints guarantees that each job appears in exactly one of the selected configurations; and each configuration of jobs must have total processing time at most τ on each machine. Since there are an exponential number of possible configurations, the number of variables in LP2 is also exponential; thus, solving LP2 in polynomial time is challenging. However, given τ, Bansal and Sviridenko [6] showed that there is a polynomial-time algorithm that either computes a solution to LP2 with value $(1 + \epsilon)\tau$ for any constant $\epsilon > 0$, or reports that there is no solution of makespan at most τ.

As with LP1, a natural step then is to round the solutions found by solving LP2. For some instance $x \in I$ of a minimization problem, let $OPT_{IP}(x)$ and $OPT_{LP}(x)$ be the values of the optimal solutions for an IP and its LP-relaxation, respectively. The *integrality gap* for a LP-relaxation is $\max_{x \in I} OPT_{IP}(x)/OPT_{LP}(x)$. The integrality gap for LP1 is 2 [20], and for $R||C_{max}$ the integrality gap of LP2 is also 2 [97]. In 2011, Svensson [96] showed that the integrality gap of LP2 is $33/17$ for the restricted assignment problem, and using the results in [6] gave a polynomial-time algorithm that can estimate the optimal makespan within a factor $33/17 + \epsilon$ for any $\epsilon > 0$. Jansen and Rohwedder [56] improved this bound to $11/6 + \epsilon$, and Jansen *et al.* [51] improved it to $5/3 + \epsilon$ when there are only two job lengths. We note that the result by

Jansen and Rohwedder was also achieved later by the same authors [58] using a (weaker) LP relaxation that has polynomial numbers of variables and constraints. While these works allow us to estimate the optimal makespan better than a factor of 2, they do not produce a schedule in polynomial time. Recently [57] showed that there is an algorithm that runs in quasi-polynomial time (i.e. the running time of the algorithm is bounded by $2^{\log n^{O(1)}}$) that produces a schedule with makespan within a factor $11/6 + \epsilon$ of the optimum for the restricted assignment problem.

Let us summarize a few other integrality gap results known with respect to LP2. Ebenlendr *et al.* [20] showed that LP2 has integrality gap 2 for the graph balancing problem with speeds. Verschae and Wiese [97] proved LP2 has integrality gap 2 for the so-called *unrelated graph balancing problem*, which holds even in the graph balancing case when every $p_{i,j} \in \{\epsilon, 1, \infty\}$ for some $\epsilon > 0$. Finally, Jansen and Rohwedder [59] proved that LP2 has integrality gap of at most 1.749 for the graph balancing problem.

Scheduling Problems with Processing Set Restrictions

For every job $J_j \in J$, the eligibility constraints $\mathscr{M}_j$ of J_j are a subset of the machines where J_j can be scheduled. In the literature, the eligibility constraints are called various different names, including processing sets. As discussed by Leung and Li [73, 74], there has been recent interest in instances of the restricted assignment problem where the processing sets have specific structure; the main appeal here is that in these cases we can evade the $3/2$-hardness lower bound known for the general problem, and yet still produce close-to-optimal solutions for meaningful scheduling problems. The study of these classes of instances is commonly referred to as $P|\mathscr{M}_j|C_{max}$ with *processing set restrictions*, and below we abbreviate each special case $(\cdot)$ as $P|\mathscr{M}_j(\cdot)|C_{max}$.

- *Inclusive* ($P|\mathscr{M}_j(inclusive)|C_{max}$). The processing sets $\mathscr{M}_1, \mathscr{M}_2, \ldots, \mathscr{M}_n$ satisfy the property that, for any two jobs $J_j, J_{j'} \in J$, either $\mathscr{M}_j \subseteq \mathscr{M}_{j'}$ or $\mathscr{M}_j \supseteq \mathscr{M}_{j'}$. Without loss of generality we can assume that the machines are linearly ordered so that for every job $J_j \in J$ there is a positive integer $1 \leq a_j \leq m$, such that each processing set $\mathscr{M}_j = \{M_{a_j}, M_{a_j+1}, \ldots, M_m\}$. We note that inclusive processing sets are also known as Grade of Service (GoS) processing sets in the literature [71].

- *Nested* ($P|\mathscr{M}_j(nested)|C_{max}$). A generalization of $P|\mathscr{M}_j(inclusive)|C_{max}$ where, for any pair of jobs $J_j, J_{j'} \in J$, only one of the following statements is satisfied: $\mathscr{M}_j \subseteq \mathscr{M}_{j'}$, $\mathscr{M}_j \supseteq \mathscr{M}_{j'}$, or $\mathscr{M}_j \cap \mathscr{M}_{j'} = \emptyset$. As an aside, nested processing sets form a laminar family, a commonly studied set system.

- *Tree-hierarchical* ($P|\mathscr{M}_j(tree)|C_{max}$). Every machine $M_i \in M$ is a vertex in an in-tree. Every $J_j \in J$ is associated with a machine M_{a_j} and the processing set $\mathscr{M}_j$ consists of the machines along the unique path from M_{a_j} to the root of the in-tree. An example of such an in-tree is given in Figure 2.1. $P|\mathscr{M}_j(inclusive)|C_{max}$ is the special case when the in-tree is a chain; an example is shown in Figure 2.2.

In Figure 2.3, we give the complexity hierarchy for all the scheduling problems described above. Recall that $P||C_{max}$ is strongly NP-hard, thus all these problems are also strongly NP-hard; however, unlike for the general restricted assignment problem, there are ρ-approximation

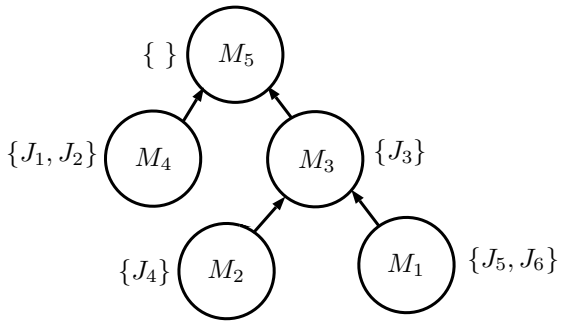

Figure 2.1: An example in-tree representing tree-hierarchical processing sets. The processing sets in this example are $\mathscr{M}_1 = \mathscr{M}_2 = \{M_4, M_5\}$, $\mathscr{M}_3 = \{M_3, M_5\}$, $\mathscr{M}_4 = \{M_2, M_3, M_5\}$, and $\mathscr{M}_5 = \mathscr{M}_6 = \{M_1, M_3, M_5\}$.

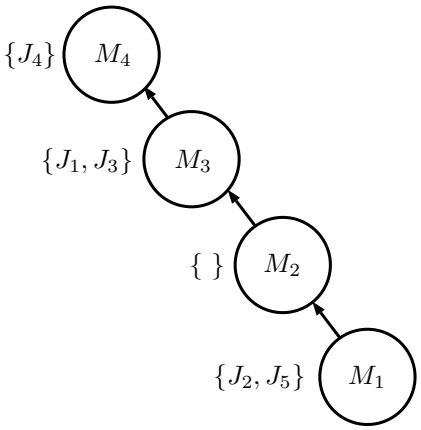

Figure 2.2: The processing sets from an instance of $P|\mathscr{M}_j(inclusive)|C_{max}$ represented as an in-tree, where $\mathscr{M}_4 = \{M_4\}$, $\mathscr{M}_1 = \mathscr{M}_3 = \{M_3, M_4\}$, $\mathscr{M}_2 = \mathscr{M}_5 = \{M_1, M_2, M_3, M_4\}$.

algorithms with approximation ratio $\rho < 2$ when the processing sets are not arbitrary. We summarize the main approximation algorithm results for these problems in Table 2.1.

2.2.3 Graph Balancing Problem ($P|\mathscr{M}_j, |\mathscr{M}_j| \leq 2|C_{max}$)

According to Asahiro *et al.* [2], minimizing the maximal load of a weighted multigraph allows one to build efficient dynamic data structures for multigraphs when the goal is to speed up the performance of vertex-adjacency operations (e.g. see [9]). Let us consider an adjacency list for a multigraph. Observe that determining if two vertices u and v are adjacent or not can be done in an adjacency list by traversing either the adjacency list for vertex v or the adjacency list for u, but checking both lists will also successfully do this. Notice that if we were to direct some edge $\{u, v\}$ in a multigraph so that u is directed toward v, the list of u decreases by one element but the list of v has the same number of elements. By directing all the edges, each vertex appears at most once among all the lists of the adjacency list and the lengths of the lists in an adjacency list can be shortened; this may quicken query times as there are less elements to traverse. Minimizing the maximal length of the lists in an adjacency list is equivalent to

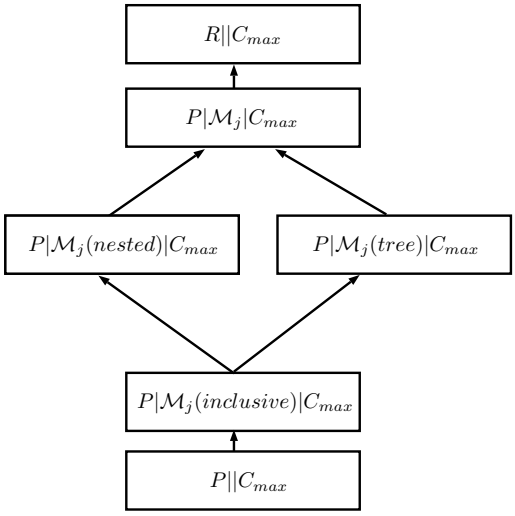

Figure 2.3: Complexity hierarchy of parallel machine scheduling problems with processing sets, where, for two scheduling problems A and B, $A \rightarrow B$ in the diagram implies A is a special case of B.

Problem	**Result**	**References**
$P\|\mathscr{M}_j(inclusive)\|C_{max}$	PTAS	Ou *et al.* [83], Li and Wang [76] (allows job release times), and Epstein and Levin [23] (speed-hierarchical model).
$P\|\mathscr{M}_j(nested)\|C_{max}$	7/4-approximation 5/3-approximation PTAS	Huo and Leung [46]. Huo and Leung [47]. Muratore *et al.* [81], and Epstein and Levin [23].
$P\|\mathscr{M}_j(tree)\|C_{max}$	4/3-approximation PTAS	Huo and Leung [46], and Leung and Ng [75] (speed-hierarchical model for trees). Epstein and Levin [46].

Table 2.1: Summary of main approximation algorithms for $P|\mathscr{M}_j|C_{max}$ with processing set restrictions.

minimizing the maximum load of the multigraph if all the edges have unit length: compute an optimal orientation then simply reverse all the directions of the edges in the orientation. See Figure 2.4 for an example. It is not hard to see that when the edges are weighted and these weights represent a priority (e.g., frequency of use), higher-priority vertices for vertex-adjacency operations can be placed in shorter lists.

Ebenlendr *et al.* [20] gave a 7/4-approximation algorithm for the graph balancing problem: This is an LP-based algorithm with an elegant threshold-based rounding scheme that increases the load of any vertex at most twice. We summarize this algorithm as we utilize ideas from it. Ebenlendr *et al.* [20] also proved that there is no ρ-approximation algorithm with $\rho < 3/2$

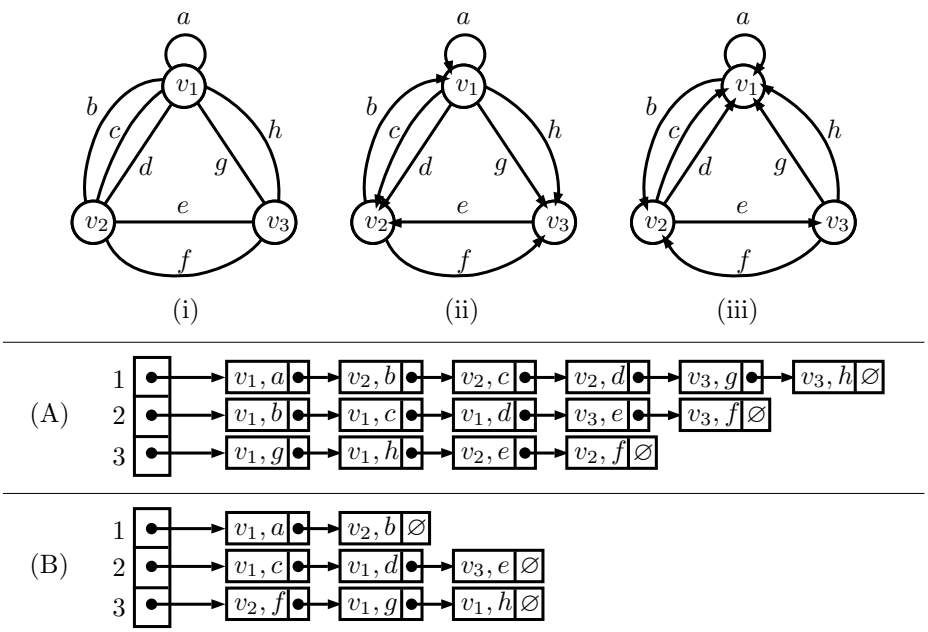

Figure 2.4: Assume we are given a multigraph G as shown in (i), every edge has length 1 and an adjacency list representation of (i) is (A). An optimal orientation of the edges in (i) is shown in (ii). Reversing the directions of the edges in (ii) we get (iii), an adjacency list of (iii) representation is (B).

for the graph balancing problem unless $\mathsf{P} = \mathsf{NP}$; we present this reduction below. The best-known approximation algorithm for the graph balancing problem has approximation ratio $7/4$; however, Wang and Sitters [99] designed a simpler $11/6$-approximation algorithm based on the LP-rounding approach of Shmoys and Tardos [93] for $R||C_{max}$.

Prior to our work, the best-known approximation algorithm for the graph balancing problem with two job lengths was a 1.652-approximation algorithm by Kolliopoulos and Moysoglou [66]. Their approximation algorithm employs binary search, and for a given estimation of the minimum makespan τ different approximation algorithms are used based on the values of the weights with respect to τ. A core component of this approximation algorithm is a flow-network based approximation algorithm for the two-valued case of the restricted assignment problem introduced in the same paper.

There also has been some work studying the graph balancing problem when the input multigraph is simple; this problem is called the graph orientation problem. We summarize some of these results later.

$7/4$-Approximation Algorithm of Ebenlendr, Krčál, and Sgall for the Graph Balancing Problem

Here we summarize the $7/4$-approximation algorithm of Ebenlendr *et al.* [20]. This algorithm uses a $7/4$-relaxed decision procedure. First, all the edge lengths and dedicated loads are divided by the makespan estimate τ; hence, in the sequel it is assumed that $\tau = 1$. Clearly if there is some edge $e \in E$ with $p_e > 1$ or some vertex $v \in V$ with dedicated load $q_v > 1$, there is

no orientation with makespan at most 1 and `FAIL` is returned.

Before we describe the algorithm, a stronger LP-formulation for the graph balancing problem is presented. An edge is called *big* if $p_e > 1/2$ and is *small* otherwise. Let E^B be the set of big edges and let $G^B = (V, E^B)$ be the subgraph of G consisting of big edges. Let T_B be the set of all possible trees consisting of big edges in G^B. As shorthand, if v is a vertex and e is an edge, $v \in e$ means "v is incident to e". For any tree T of G, let $L(T) = \{(v, e) \in V \times E \mid v \text{ is a leaf of } T, v \in e, \text{and } e \in T\}$, where each (v, e) is called a *leaf pair* of T. Consider any big tree $T \in T_B$. Assuming $OPT \leq 1$, since T has one more vertex than edges, at most one edge in the set of leaf pairs can be oriented away from its leaf. This leads to what is called the *tree constraint* (Tree T) in linear program LP3 shown below. A variable x_{ev} is defined for each edge $e \in E$ and endpoint v of e. If $x_{ev} = 1$, e is oriented towards v. If $0 < x_{ev} < 1$, we say that e is *fractionally oriented* towards v.

Graph Balancing Linear Program 1 (LP3)

$$
\begin{aligned}
x_{eu} + x_{ev} &= 1, && \text{for all } e = \{u, v\} \in E && \text{(Edge } e\text{)} \\
q_v + \sum_{e \mid v \in e} p_e x_{ev} &\leq 1, && \text{for all } v \in V && \text{(Load } v\text{)} \\
\sum_{(v,e) \in L(T)} p_e x_{ev} &\geq \sum_{(v,e) \in L(T)} p_e - 1, && \text{for each } T \in T_B && \text{(Tree } T\text{)} \\
x_{ev} &\geq 0, && \text{for all } e \in E, v \in e, &&
\end{aligned}
$$

where the set of leaf pairs of tree T is

$$L(T) = \Big\{(v, e) \mid v \text{ is a leaf of } T, e \in T\Big\}.$$

Adding the tree constraints makes LP3 stronger than LP1. LP3 can have exponentially-many constraints as there can be exponentially-many big trees, so Ebenlendr *et al.* showed that solving LP4 below yields a feasible solution to LP3; LP4 has a polynomial number of constraints. We dedicate more discussion to LP3 and LP4 in Chapter 4.2.3, where we introduce generalizations of these LPs. Note that both LP3 and LP4 have integrality gap 7/4 for the graph balancing problem.

Graph Balancing Linear Program 2 (LP4)

$$
\begin{aligned}
x_{eu} + x_{ev} &= 1, && \text{for all } e = \{u, v\} \in E && \text{(Edge } e\text{)} \\
q_v + \sum_{e \mid v \in e} p_e x_{ev} &\leq 1, && \text{for all } v \in V && \text{(Load } v\text{)} \\
\sum_{\substack{e \mid v \in e, \\ p_e > 1/2}} x_{ev} &\leq 1 && \text{for all } v \in V && \text{(Star } v\text{)} \\
x_{ev} &\geq 0, && \text{for all } e \in E \text{ and } v \in e, &&
\end{aligned}
$$

Now we can present the 7/4-relaxed decision algorithm. First, solve LP4 to obtain a feasible solution $\mathbf{x}$ to LP3. Let $E_x = \{e \in E \mid 0 < x_{eu} < 1 \text{ for } u \in e\}$. Also, let $G_x = (V, E_x)$ and

$G_x^B = (V, E_x \cap E^B)$. If a feasible solution is not found, then $OPT > 1$ and return FAIL. The algorithm then considers fractionally oriented edges in G_x, and performs the rounding procedure described below to determine their final orientations. It is assumed that as edges are oriented, G_x and G_x^B are updated accordingly. If there is a vertex v of degree 1 in G_x (i.e., there is a leaf pair (v, e)) and $0 < x_{ev} < 1$ for edge $e = \{u, v\}$ then

- *Leaf assignment:* If $p_e x_{eu} \leq 3/4$, e is oriented towards the leaf v (see Figure 2.5);

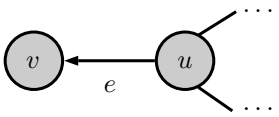

Figure 2.5: An example of a leaf assignment. Note that the edge oriented in a leaf assignment can be big or small.

- *Tree assignment:* If $p_e x_{eu} > 3/4$ note that e then must be a big edge and the connected component of G_x^B containing e must be a tree T. Orient all edges in T away from v (see Figure 2.6).

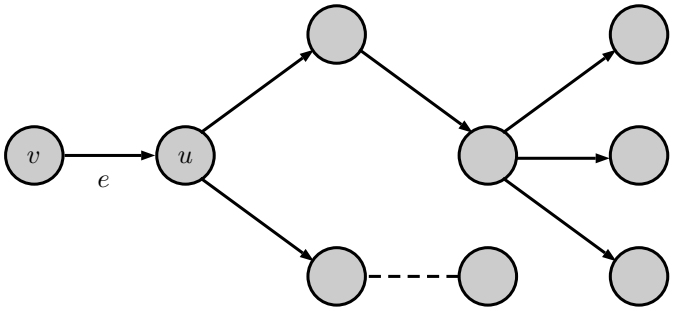

Figure 2.6: An example of edges being oriented in a tree assignment. The solid lines form the tree T of big edges. The dashed lines represent small edges, which are not oriented by the tree assignment.

If no vertex v as above is found then there must be a cycle C, we then perform a *rotation*. Traverse G_x to find a cycle selecting big edges over small ones. Once a cycle C is found, apply ROTATE($\mathbf{x}, C$): Compute $\delta \leftarrow \min_{e=(u,v)\in C}(p_e x_{eu})$ and then for each $e = (u, v) \in C$, set $x_{eu} \leftarrow x_{eu} - \delta/p_e$ and $x_{ev} \leftarrow x_{ev} + \delta/p_e$. After performing ROTATE($\mathbf{x}, C$), at least one edge $e = \{u, v\}$ in C has $x_{eu} = 0$ and $x_{ev} = 1$ so e is oriented to v. Observe that rotations do not change the loads of vertices. The algorithm terminates once G_x has no edges.

The rounding performed by a leaf assignment increases the load of a vertex u by at most $3/4$ if the edge e under consideration is big or it increases by at most $1/2$ if e is small. Furthermore, a tree assignment can increase the load of a vertex u by at most $1/4$. A vertex u can have its load increased by either (i) only one leaf assignment or (ii) by a tree assignment plus a leaf assignment involving a small edge. In either case the maximum load of a vertex is at most $7/4$.

3/2-Hardness of the Graph Balancing Problem

Here we present the 3/2-hardness lower bound for the graph balancing problem, even when the jobs have lengths either 1 or 2. The reduction we present is by Ebenlendr *et al.* [21]; this reduction utilises a variant of the satisfiability problem called At-most-3-SAT(2L). At-most-3-SAT(2L) is known to be NP-complete [2]. We note that a similar reduction was also independently given by Asahiro *et al.* [2].

Problem: At-Most-3-SAT(2L)

Input: A propositional logic formula ϕ in conjunctive normal form (CNF), where there are n' boolean variables $x_1, \ldots, x_{n'}$ and m' clauses $y_1, y_2, \ldots, y_{m'}$. There are at most three literals per clause, each variable appears at most three times in ϕ, and each literal (a variable or its negation) occurs at most twice in ϕ.

Output: Is there an assignment of values to the variables $x_1, \ldots, x_{n'}$ such that ϕ is satisfied?

Given an instance of At-Most-3-SAT(2L), we build a weighted multigraph G as follows; assume all vertices have zero dedicated load unless otherwise stated. Create one vertex for each clause y_i, and one vertex for each literal (x_i and $\neg x_i$) of every variable x_i; let the former be called *clause vertices* and the latter be called *literal vertices*. For each variable x_i, add an edge $\{x_i, \neg x_i\}$ with weight 2 called a *tautologous edge*; the dedicated load of clause vertex y_i is $3 - |y_i|$, where $|y_i|$ is the number of literals in clause y_i. Finally, for each clause y_i and literal l, add a *clause edge* $\{l, y_i\}$ if literal l is in clause y_i.

To illustrate the reduction, we give an example. Let the propositional logic formula $\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$, where $n' = 3$ and $m' = 2$. Then $y_1 = (x_1 \vee \neg x_2)$ and $y_2 = (\neg x_1 \vee \neg x_2 \vee \neg x_3)$. Applying the reduction we obtain the multigraph G shown in Figure 2.7. The formula ϕ can be satisfied and G has an optimal orientation with maximal load 2.

Theorem 2.2.2 (Ebenlendr *et al.* [21]) *There is an orientation of G with makespan at most 2 if and only if ϕ can be satisfied.*

Proof To prove the claim, we consider each direction separately.

($\Rightarrow$) Given an orientation of G with makespan at most 2, set each variable x_j as follows: Set $x_j = 1$ if tautologous edge $\{x_j, \neg x_j\}$ is oriented toward literal vertex $\neg x_j$, and $x_j = 0$ otherwise.

For any clause y_i, there must be at least one literal l (for some variable $x_{j'}$) such that its clause edge $\{l, y_i\}$ is directed toward literal vertex l; otherwise, the load of clause vertex y_i exceeds 2 since the dedicated load of y_i is $3-|y_i|$. Note that all other clause edges incident on clause vertex y_i can be directed toward y_i without the load exceeding 2. Then, the tautologous edge $\{x_{j'}, \neg x_{j'}\}$ is not oriented toward literal vertex l since the length of any tautologous edge is 2. Thus, the orientation of clause edge $\{l, y_i\}$ toward l corresponds to setting a value for $x_{j'}$ so that y_i is satisfied, and ϕ is satisfied.

($\Leftarrow$) Given an assignment of values to variables $x_1, x_2, \ldots, x_{n'}$, construct an orientation in the following way: For each variable x_j, direct tautologous edge $\{x_j, \neg x_j\}$ toward literal

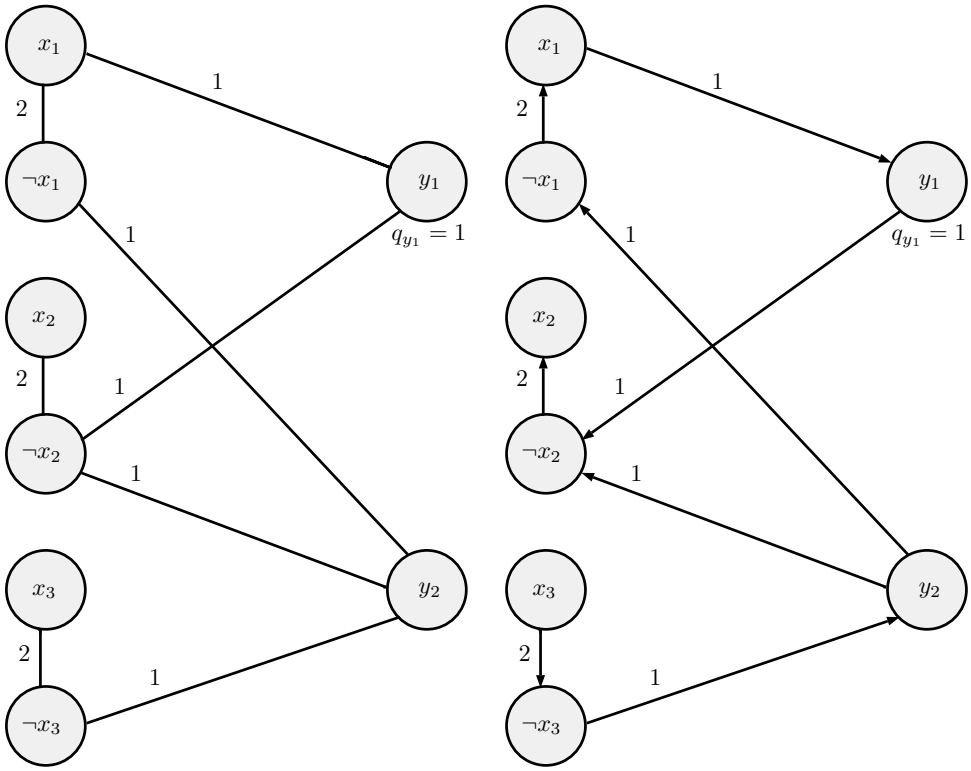

Figure 2.7: Given the formula $\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$, the resulting graph balancing instance applying the above construction is shown on the left. Its optimal orientation is given on the right.

vertex x_j if $x_j = 0$, and toward literal vertex $\neg x_j$ if $x_j = 1$. Next, each clause edge $\{l, y_i\}$ of clause y_i and literal l is oriented toward literal vertex l if l is true, and toward y_i if l is false.

Consider any literal vertex l. If literal l is false then a tautologous edge of length 2 is directed toward literal vertex l, and if l is true then at most two clause edges, each of length 1, are directed toward l. So the load of any literal vertex is at most 2. Since every clause y_i is satisfied, at least one clause edge is directed away from y_i and the load of clause vertex y_i is at most $(3 - |y_i|) + (|y_i| - 1) \leq 2$. ∎

By Theorem 2.2.2, G has an orientation with makespan at most 2 if ϕ is satisfied, but the makespan is at least 3 otherwise. Hence, if there were a ρ-approximation algorithm with $\rho < 3/2$, one could apply the above reduction and the ρ-approximation algorithm, then correctly decide whether ϕ is satisfiable or not in polynomial time: If the makespan is less than 3 return "yes"; otherwise, return "no".

Corollary 2.2.3 (Ebenlendr *et al.* [21]) *There is no ρ-approximation algorithm with $\rho < 3/2$ for the graph balancing problem with job lengths either 1 or 2, unless* P = NP.

Graph Orientation Problem ($P|\mathscr{M}_j, |\mathscr{M}_j| = 2|C_{max}$)

There has been some attention given to the graph balancing problem when the input multigraph is simple. That is, no more than one edge exists between two common endpoints, and there are no self loops (i.e., the dedicated loads $q_v = 0$). This case is referred to in the literature as the *graph orientation problem*, and for this problem the 3/2-hardness lower bound still persists [2] using a reduction similar to the one given previously. This problem was originally proposed by Asahiro *et al.* [4], where the goal is to find an orientation for the edges so that the maximum weighted outdegree of the vertices is minimized. It is not hard to see minimizing the maximum weighted outdegree is equivalent to minimizing the maximum load of a vertex (the maximum weighted indegree), by simply reversing the orientation of edges in an orientation.

Currently much of the contributions associated with this problem are found in papers by Asahiro *et al.* [2, 3, 4]. The best-known approximation algorithms for the graph orientation problem are those for the graph balancing problem. Asahiro *et al.* [2] gave a combinatorial $(2-\ell_s/\ell_b)$-approximation algorithm for the graph orientation problem with two job lengths $p_e \in \{\ell_s, \ell_b\}$, $\ell_s < \ell_b$, which is tight with the 3/2 lower bound when $\ell_s = 1$ and $\ell_b = 2$. It is also worth noting that the graph orientation problem is polynomial-time solvable on trees and cacti, it is weakly NP-hard on series-parallel graphs, and it is strongly NP-hard for planar bipartite graphs [3].

2.3 Other Scheduling Problems

2.3.1 Scheduling Parallel Machines with Simple Job-Intersection Structure and Bounded Job Assignments

The concept of the job-intersection graph goes back to at least Glass and Kellerer [33] with the study of so-called nested-structures (Chapter 2.2.2) and the restricted assignment problem. Research on the restricted assignment problem when instances satisfy certain structural properties is extensive and has grown in interest in recent years [74]. In addition, there has been investigation of scheduling problems on machine-intersection graphs where the machines are the vertices and an edge exists between two vertices when a job can be scheduled on the two corresponding machines [3, 53, 70]. Jansen *et al.* [53][1] proved that $R||C_{max}$ is fixed-parameter tractable (FPT) in the treewidth tw of the job-intersection graph. That is, if the job-intersection graph G_J has constant treewidth, $R||C_{max}$ can be solved in polynomial time. So when the job-intersection graph belongs to graph classes such as trees ($tw = 1$), cactus graphs ($tw \leq 2$), outerplanar graphs ($tw \leq 2$), and series-parallel graphs ($tw \leq 2$), $R||C_{max}$ is solvable in polynomial time. In Figure 2.8 we summarize both, computational complexity results found in this thesis and results presently in the literature for $R||C_{max}$ with simple job-intersection structure.

For any $0 \leq r \leq n$, it is trivial to determine if the set $\mathscr{J}_i$ of jobs that every machine M_i can process has size at most r. Alon *et al.* [1] gave an algorithm that can test if a graph (V, E) is triangle free in $O(|E|^{1.41})$ time. We note that diamondless graphs can be recognized in $O(|V|^3)$ time by a simple algorithm that looks for a pair of triangles with a common edge. Kloks *et al.* [64] showed that one can recognize if a graph is diamond-free (and give a diamond in the

[1] In this paper the authors refer to the job-intersection graph as the primal graph.

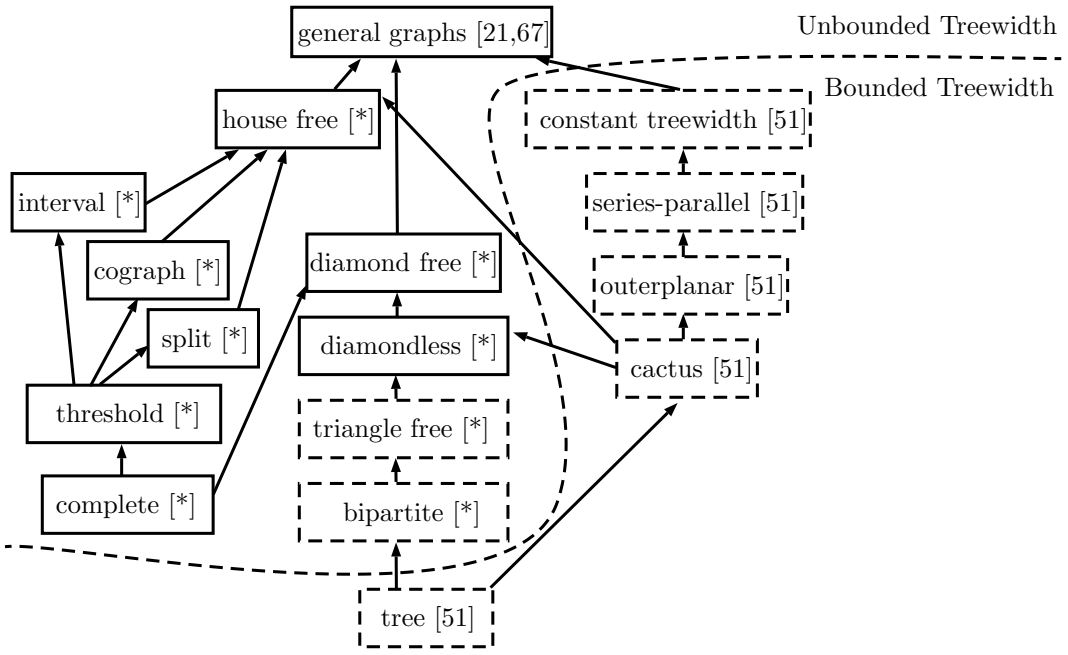

Figure 2.8: Summary of results for $R||C_{max}$ with simple job-intersection structure. The job-intersection graphs restricting the machine assignments are grouped by graph class. For two graph classes A and B, "$A \rightarrow B$" in the diagram means that any graph in A is in graph class B. Problems boxed with dashed lines are polynomial-time solvable, and problems with boxed solid lines are strongly NP-hard. The number(s) in brackets are reference numbers, and graph classes with [*] beside them refer to computational complexity results found in this thesis.

graph if it is not) in $O(|V|^c + |E|^{3/2})$ time, where $O(|V|^c)$ is the time complexity to compute the square of a $|V| \times |V|$ 0-1 adjacency matrix.

To the best of our knowledge $R||C_{max}$ with bounded job assignments has not been previously studied. Bounded job assignments have been considered in other types of scheduling problems, such as in batch scheduling where a batch size bounds the number of jobs simultaneously processed by a batching machine [12, 78]. A generalization of $R||C_{max}$ where every machine has a positive integer called a *machine capacity* that bounds the maximum number of jobs each machine can process has also been studied. For this generalization there is a 2-approximation algorithm [90], and there exists an efficient polynomial-time approximation scheme when the machines are identical [14].

2.3.2 Scheduling with Machine Types

Recently, Jansen and Maack [52] presented an EPTAS for $R||C_{max}$ with machine types when the number of machine types is constant. Prior to this result, Gehrke *et al.* [31] gave a PTAS for $R||C_{max}$ when the number δ of machine types is constant, its running time is bounded by

$$O(\delta n) + m^{O(\delta/\epsilon^2)} \cdot \left(\frac{\log m}{\epsilon}\right)^{O(\delta^2)}.$$

In addition, Knop and Koutecký [65] showed that $R||C_{max}$ is fixed-parameter tractable in parameters $\max_{i,j} p_{i,j}$ and δ, i.e. there is an algorithm with running time bounded by $f(\delta, \max_{i,j} p_{i,j}) \cdot poly(|x|)$ that optimally solves $R||C_{max}$, where f is some computable function and $|x|$ is the input size of instance x. For some more discussion on results for special cases of makespan minimization with machine types see [31, 52].

In a similar vein, two jobs J_j and $J_{j'}$ have the same *job type* if $p_{i,j} = p_{i,j'}$ for all $M_i \in M$. Goemans and Rothvoß [34] showed that $P||C_{max}$ is polynomial-time solvable when the number of job types is constant. Later, Chen *et al.* [15] generalized upon this work to prove that $R||C_{max}$ is polynomial-time solvable if both the number of job types and the number of machine types are bounded by a constant.

2.3.3 Scheduling Parallel Machines with Bags

Makespan minimization with bags is a type of conflict scheduling problem, where two jobs from the same bag conflict if they are scheduled on the same machine. A natural way to model this type of conflict is with an incompatibility graph: There is a vertex for each job and an edge $\{J_j, J_{j'}\}$ if jobs J_j and $J_{j'}$ cannot be scheduled on the same machine. Then, makespan minimization with ℓ bags is when the incompatibility graph consists of ℓ disjoint cliques. Bodlaender *et al.* [7] developed several results for $P||C_{max}$ with incompatibility graphs. Even *et al.* [24] studied a similar conflict scheduling problem where two conflicting jobs cannot be scheduled concurrently. In addition, Dokka *et al.* [19] considered a related, but generalized version of $P|bag|C_{max}$ called the multi-level bottleneck assignment problem, and gave a 2-approximation algorithm for three bags.

In 2017, Das and Wiese [17] presented a PTAS for $P|bag|C_{max}$, and an 8-approximation algorithm for the restricted assignment problem with bags in the special case when for each bag B_k all the jobs $j \in B_k$ have the same eligibility constraints. In addition, for any $\epsilon > 0$, they proved there is no $((\log n)^{1/4-\epsilon})$-approximation algorithm for the restricted assignment problem with bags, unless $\mathsf{NP} \subseteq \mathsf{ZPTIME}(2^{(\log n)^{O(1)}})$[2]. They accomplished this by reducing from vector scheduling, which is known to have no constant-factor approximation algorithm [13], unless $\mathsf{NP} = \mathsf{ZPP}$[3]. As pointed out by Das and Wiese, by applying the results of Zuckerman [103] to [13], the same reduction implies that there is no constant-factor approximation algorithm for the restricted assignment problem with bags, unless $\mathsf{P} = \mathsf{NP}$. Recently Grage *et al.* [36] improved upon the PTAS of Das and Wiese by presenting an EPTAS for $P|bag|C_{max}$. To the best of our knowledge, $R|bag|C_{max}$ with a small number of bags has not yet been considered, nor when all the jobs have unit length.

2.3.4 Preemptive Scheduling

In this thesis we focus on non-preemptive scheduling problems. That is, once a job begins being processed it must continue being executed until it completes. In contrast, in preemptive scheduling jobs may be interrupted and executed later; each time a job is interrupted is called a *preemption*. It is assumed that there is no cost associated with a preemption.

[2] $\mathsf{ZPTIME}(f(n))$ - Zero-error probabilistic $f(n)$-time.

[3] ZPP - Zero-error probabilistic polynomial time.

Lawler and Labetoulle [69] showed that makespan minimization on unrelated parallel machines where the jobs are preemptively scheduled ($R|pmtn|C_{max}$) is polynomial-time solvable; their algorithm performs no more than $O(m^2)$ many preemptions. There are two steps in this algorithm. First, the algorithm solves the following LP: τ is a variable representing the makespan and the objective is to minimize τ. Let $t_{i,j}$ be a variable that represents the total amount of time that a machine M_i spends executing job J_j, where $t_{i,j} \geq 0$ for each $M_i \in M$ and $J_j \in J$; then the LP has constraints $\sum_{i=1}^{m} t_{i,j}/p_{i,j} = 1$ for each job $J_j \in J$ (each job needs to be processed entirely), $\sum_{i=1}^{m} t_{i,j} \leq \tau$ for each job $J_j \in J$ (no job can process longer than τ over all the machines), and $\sum_{j=1}^{n} t_{i,j} \leq \tau$ for $M_i \in M$ (each machine has total processing at most τ). Then, the algorithm constructs an optimal schedule from an optimal solution to this LP by solving an instance of the so-called preemptive open shop scheduling problem (referred to as $O|pmtn|C_{max}$ in the literature); this can be accomplished in $O(r^2)$-time using the algorithm of Gonzalez and Sahni [35], where r is the number of variables with $t_{i,j} > 0$; note that $r \leq mn$.

The algorithm described above is not very fast as it requires solving a linear program with $O(m + n)$ constraints and $O(mn)$ variables. Faster FPTASs for $R|pmtn|C_{max}$ have since been presented [55, 88].

2.3.5 Precedence-Constrained Scheduling

In this thesis we consider the setting where jobs can be processed in any order, but for completeness here we discuss some results for scheduling problems when this is not true. In precedence-constrained scheduling, the order in which jobs must be processed is given as a directed acyclic graph (DAG), where the jobs are vertices in the graph and an arc $(J_j, J_{j'})$ implies job J_j must complete before job $J_{j'}$ can start. Clearly the case when there are no precedence constraints is a special case, as in its DAG is every J_j is isolated.

The literature in this area is extensive, thus we only briefly discuss a few results here. On identical parallel machines, an algorithm of Graham [37] is a $(2 - 1/m)$-approximation algorithm for $P|prec|C_{max}$. On the other hand, there is no ρ-approximation algorithm with constant $\rho < 2$ even when the jobs have unit length ($P|p_j = 1, prec|C_{max}$), unless a stronger variant of the unique games conjecture (see Bansal and Khot [5]) holds [95]. On uniform parallel machines, Chudak and Shmoys [16] gave a $O(\log m)$-approximation algorithm for $Q|prec|C_{max}$ and recently Li [77] presented a $O(\log m/ \log \log m)$-approximation algorithm. In the case of unrelated parallel machines, not as much is known for $R|prec|C_{max}$ (see e.g. [68]).

Chapter 3

Graph Balancing Problem with Two Job Lengths

In this chapter we present two approximation algorithms for the graph balancing problem with two job lengths ℓ_s and ℓ_b, where $\ell_s, \ell_b \in \mathbb{Z}^+$ and $\ell_s < \ell_b$; note that each dedicated load of a vertex $v \in V$ is a linear combination of ℓ_s and ℓ_b, i.e. $q_v = a\ell_s + b\ell_b$, for $a, b \in \mathbb{Z}^+ \cup \{0\}$. The first algorithm presented in Chapter 3.1 relies on a combination of flow theory, matchings, and linear programming, while the latter given in Chapter 3.2 is considerably simpler and is purely combinatorial. As a full record of research completed, both algorithms are included. Both of the approximation algorithms we present use $3/2$-relaxed decision procedures (Chapter 1.1.4).

3.1 First $3/2$-Approximation Algorithm

The $3/2$-relaxed decision algorithm GB2W (Algorithm 1) presented below computes a schedule with makespan at most $3\tau/2$ if a schedule with makespan at most τ exists. If no schedule with makespan at most τ exists, the algorithm returns `FAIL`.

Algorithm 1: GB2W $(G = (V, E, \mathbf{p}, \mathbf{q}), \tau)$

Input: Multigraph G, value τ.
Output: An orientation γ for the edges in E with makespan at most $3\tau/2$ or `FAIL`. If `FAIL` is returned there is no orientation with makespan τ.

1. Divide ℓ_s, ℓ_b, edge lengths, and vertex dedicated loads by τ; set $\tau = 1$.
2. **If** $\ell_b > 1$ or $q_v > 1$ for any $v \in V$ **then return** `FAIL`.
3. **If** $\ell_s, \ell_b \in (0, 1/2]$ **then** apply Theorem 2.2.1.
4. **If** $\ell_s, \ell_b \in (1/2, 1]$ **then** apply the algorithm given in Lemma 3.1.1.
5. **If** $\ell_s \in (0, 1/2]$ and $\ell_b \in (1/2, 1]$ we consider three subcases.
   Let $k = \lfloor 1/\ell_s \rfloor$, so $1/(k+1) < \ell_s \leq 1/k$.
   5.1. **If** $k/(k+1) \leq \ell_b \leq 1$ **then** apply the algorithm in Lemma 3.1.2.
   5.2. **If** $(k-1)/k \leq \ell_b < k/(k+1)$ **then** apply the algorithm in Lemma 3.1.3.
   5.3. **If** $1/2 < \ell_b < (k-1)/k$ **then** apply the algorithm in Lemma 3.1.4.
6. **If** any of the algorithms used in Steps 3–5 reports `FAIL` or if the solution computed by them has value larger than $3/2$ **then return** `FAIL`; otherwise **return** the solution computed in the above steps.

The remainder of this section gives Lemmas 3.1.1–3.1.4 which provide the subroutines used by our algorithm, then in Theorem 3.1.6 we prove our algorithm is a 3/2-approximation algorithm for the graph balancing problem with two job lengths. First, we present Lemma 3.1.1 which covers Step 4 of our algorithm. We define a *big edge e* to be an edge with length $p_e > 1/2$; otherwise, we call an edge *small*. From this point forward, we assume that as edges are oriented in Steps 4–5.3, they are removed from G.

3.1.1 Step 4

Note that in Step 1 the algorithm scales the job lengths so the estimation for the optimum makespan is $\tau = 1$.

Lemma 3.1.1 *There is a polynomial-time algorithm for the graph balancing problem with two rational lengths $\ell_s, \ell_b \in (1/2, 1]$ that finds a solution of value at most 1 if OPT ≤ 1.*

Proof Since $\ell_s, \ell_b \in (1/2, 1]$, each edge of the input multigraph G is a big edge. If there is an orientation for the edges with maximal load at most 1, then at most one edge can be oriented towards a given vertex. Therefore, if any connected component $C = (V_C, E_C)$ of G has $|E_C| > |V_C|$ then there is no solution for the graph balancing problem of value at most 1. The algorithm to compute an orientation with makespan at most 1 for the graph balancing problem with two lengths $\ell_s, \ell_b \in (1/2, 1]$ is as follows (Big_$\ell_s\ell_b$, Algorithm 2).

Algorithm 2: Big_$\ell_s\ell_b$ ($G = (V, E, \mathbf{p}, \mathbf{q})$)

Input: Multigraph G.

Output: An orientation γ for the edges in E with maximum vertex load at most 1 or `FAIL`. If `FAIL` is returned there is no orientation for E with makespan at most 1.

1. **If** any connected component $C = (V_C, E_C)$ of G has $|E_C| > |V_C|$, **then return** `FAIL`.
2. **While** G has cycles **do**
   - 2.1. Find a cycle C' of G.
   - 2.2. Mark the vertices in C' and orient the edges in C' in the same arbitrary direction along the cycle. **If** C' contains any vertex with $q_v > 0$, **then return** `FAIL`.
   - 2.3. **else** remove the edges in C' from G.
3. **For** every maximal tree T in G **do**
   - 3.1. **If** there is a vertex v in T with $q_v > 0$ or that is marked, **then** set v as the root of T
   - 3.2. **else** choose any vertex v in T as the root.
   - 3.3. Orient all the edges in T away from its root.
   - 3.4. **If** any edge in T is oriented towards a vertex u with $q_u > 0$ or that is marked **then return** `FAIL`.
4. **Return** orientation for the edges of G.

Multigraph G must have at most $|V|$ edges, and an optimal orientation matches each edge to a unique vertex with no dedicated load. As each connected component C has $|E_C| \leq |V_C|$,

C contains at most one cycle. Thus, if Step 2 finds a cycle containing a vertex v with $q_v > 0$, no orientation with makespan at most 1 exists. Next, in Step 3 the algorithm selects a vertex v as the root, then the edges in T are oriented away from v. If T does not contain any marked vertex nor a vertex v with $q_v > 0$, then any vertex v in T is selected as the root and all the edges are directed so that the load of each vertex in T is at most 1. If the root v is marked or has $q_v > 0$ and the condition of Step 3.4 is satisfied, then it must be that either an edge is directed toward a vertex u in T with $q_u > 0$ or a marked vertex, implying there must be no orientation with makespan at most 1. In the case when a root v is marked or has $q_v > 0$ and the condition of Step 3.4 is not satisfied, then every edge in T is oriented toward a vertex u that is both unmarked and has $q_u = 0$. Therefore, if a solution of value at most 1 exists, it must be found when Step 4 is reached. The above algorithm runs in $O(|V| + |E|)$ time. ∎

3.1.2 Step 5.1

We consider here the case handled in Step 5.1 of the algorithm, namely when $k/(k+1) \leq \ell_b \leq 1$, where $k = \lfloor 1/\ell_s \rfloor$ and so $1/(k+1) < \ell_s \leq 1/k$. In Lemma 3.1.2 below, we utilize the property that $\ell_s + \ell_b > 1$, which means that if $OPT \leq 1$ then no small edge can be oriented with a big edge toward the same vertex.

Lemma 3.1.2 *For any integer $k \geq 2$, there is a polynomial-time algorithm for the graph balancing problem with two rational lengths ℓ_s, ℓ_b, where $1/(k+1) < \ell_s \leq 1/k$ and $k/(k+1) \leq \ell_b \leq 1$ that finds a solution of value at most $3/2$ if $OPT \leq 1$.*

Proof If $OPT \leq 1$, since $\ell_s + \ell_b > 1$ for any optimal orientation either at most one big edge is oriented towards a vertex, or at most k small edges are oriented towards a vertex. To find an orientation for the edges of the input multigraph $G = (V, E, \mathbf{p}, \mathbf{q})$, we use a flow network N similar to one used by Kolliopoulos and Moysoglou [66].

The flow network N is built as follows. First, we consider the dedicated loads of the vertices. For each vertex $v \in V$, define a value $\beta_v \geq 0$ as follows: if $q_v \geq \ell_b$, set $\beta_v = k$; otherwise set β_v equal to the unique non-negative integer such that $q_v = \beta_v \ell_s$, assign $\beta_v = p_v$. The flow network N has a source s^* and sink t^*. We will describe the network level by level. First, we have a level of nodes in the network called *edge nodes*. These are nodes corresponding to the edges in G. There are two types of edge nodes: big edge nodes for edges with length ℓ_b; and small edge nodes for those with length ℓ_s. Add an arc from s^* to each edge node, and set its capacity to k if it is to a big edge node, and 1 otherwise.

The second level consists of *buffer nodes*, one for each vertex. The buffer nodes are added to prevent excess flow from being carried through the big edge nodes. While this part of the network is not important for proving the lemma, it will be vital for how we later use this network in Lemma 3.1.3. For each big edge $\{u, v\} \in E$, add arcs with capacity k from its big edge node to buffer nodes u and v.

For the next level of the network, create a *vertex node* that corresponds to each vertex in G. Add an arc from the buffer node of each vertex v to its vertex node with capacity k. For each small edge $\{u, v\} \in E$, include arcs with capacity 1 from its small edge node to vertex nodes u and v. Finally, for each vertex $v \in V$, add an arc from vertex node v to sink t^* with capacity $k - \beta_v$. The resulting flow network is shown in Figure 3.1.

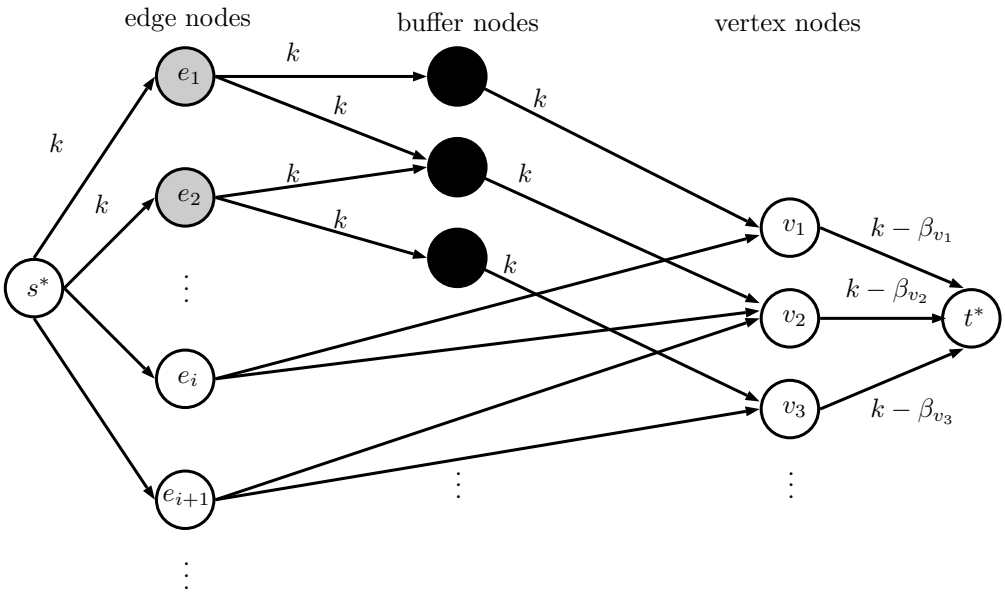

Figure 3.1: The flow network N for the case when $1/(k+1) < \ell_s \leq 1/k$ and $k/(k+1) \leq \ell_b \leq 1$. Shaded nodes represent big edge nodes, and each black node is a buffer node associated with one vertex node. Arcs that are unlabelled have flow capacity 1.

Below we give the algorithm.

Algorithm 3: BigSmall_$\ell_s\ell_b$ ($G = (V, E, \mathbf{p}, \mathbf{q})$)

Input: Multigraph G.

Output: An orientation γ for the edges in E with makespan at most 3/2 or `FAIL`. If `FAIL` is returned there is no orientation for E with makespan at most 1.

1. Build the flow network N. Set $k = \lfloor 1/\ell_s \rfloor$.
2. Compute an integral maximum flow f of N.
3. **If** all the arcs leaving the source are not saturated in f, **then return** `FAIL`.
4. Construct bipartite graph $G' = (V_{big} \cup V_{rec}, E')$, where V_{big} is the set of big edge nodes, V_{rec} is the set of buffer nodes that receive at least $\lceil k/2 \rceil$ units of flow from a big edge node, and
$$E' = \{(u, v) \mid u \in V_{big}, v \in V_{rec}, f(u, v) \geq \lceil k/2 \rceil\}.$$
5. Compute a matching $\mathscr{M}$ of G' that matches nodes in V_{big} with nodes in V_{rec}.
6. **For each** arc (u, u_i) in $\mathscr{M}$, orient big edge u towards vertex u_i.
7. **For each** small edge node u and vertex node u_i with $f(u, u_i) = 1$, orient u towards u_i.
8. **Return** orientation for the edges of G.

Both the flow network N and the bipartite graph G' constructed by the algorithm consist of $O(|V| + |E|)$ nodes and arcs. In addition, computing an integral maximum flow can be done in $O((|V| + |E|)^2)$ time using Orlin's algorithm [82], and the matching in Step 5 can be computed in $O((|V| + |E|)^{3/2})$ time using the algorithm of Hopcroft and Karp [44]. Therefore, the time

complexity of algorithm BigSmall_$\ell_s\ell_b$ (Algorithm 3) is polynomial.

Now we prove that this algorithm finds an orientation with maximal load at most 3/2 if $OPT \leq 1$. First we show that flow network N has the following property: If $OPT \leq 1$, an integral maximum flow on this network saturates all the arcs leaving s^*. Consider any optimal orientation γ^*. We construct a flow function in which every small edge node receives 1 unit of flow, and each big edge node receives k units of flow from the source s^*. For each big edge $\{u_1, u_2\}$ of G, if u is the big edge node for $\{u_1, u_2\}$, send k units of flow from u to buffer node u_i if $\gamma^*(u_1, u_2) = u_i$, k units of flow from buffer node u_i to vertex node u_i, and k units of flow from u_i to the sink t^*. Similarly, for each edge $\{u_1, u_2\}$ represented by small edge node u, if $\gamma^*(u_1, u_2) = u_i$, send 1 unit of flow from u to vertex node u_i, and one unit of flow from u_i to t^*. It is not hard to see that this flow function is feasible and that no additional flow can be sent through the network.

If $OPT \leq 1$, then as shown above, every small edge node receives 1 unit of flow from s^*. By flow conservation, each small edge node u sends its one unit of flow to a vertex node u_i, and the algorithm orients small edge u towards vertex u_i. As a result, every small edge is oriented by the algorithm. What remains to be shown is that all the big edges are oriented. The orientation of each big edge is determined by the matching $\mathscr{M}$ computed by the algorithm. We must show that $\mathscr{M}$ covers all big edge nodes. Consider any subset $V'_{big} \subseteq V_{big}$, and denote the neighbourhood in G' of this subset of nodes as $N_{G'}(V'_{big})$. Note that $N_{G'}(V'_{big}) \subseteq V_{rec}$. To show $\mathscr{M}$ exists, we use Hall's Theorem [41], which requires us to prove that $|V'_{big}| \leq |N_{G'}(V'_{big})|$. Two key observations are that the outdegree of every edge node is 2, and every big edge node receives at most k units of flow. Furthermore, by flow conservation every big edge node must send at most k units of flow to the buffer nodes. We have two cases:

- If k is odd, a buffer node in $N_{G'}(V'_{big})$ can receive at least $\lceil k/2 \rceil$ units of flow from only one big edge node in V'_{big} because $k < 2 \cdot \lceil k/2 \rceil$. Furthermore, since $k - \lceil k/2 \rceil = \lfloor k/2 \rfloor < \lceil k/2 \rceil$, each big edge node in G' has degree 1. Hence, $|V'_{big}| = |N_{G'}(V'_{big})|$.

- If k is even, then $\lceil k/2 \rceil = k/2$, and so a buffer node can receive $k/2$ units of flow from at most two big edge nodes. Partition $N_{G'}(V'_{big})$ into two disjoint sets N_1 and N_2, where N_1 contains the buffer nodes that receive more than $k/2$ units of flow from a big edge node, and N_2 has the buffer nodes that receive $k/2$ units of flow from a big edge node. Similar to when k is odd, each big edge node in V'_{big} is adjacent to only one buffer node in N_1, and so each buffer node in N_1 has degree 1 in G'. This leaves $|V'_{big}| - |N_1|$ vertices adjacent to buffer nodes in N_2. The indegree of each buffer node in N_2 is at least one (and no more than 2), but the outdegree of every big edge node adjacent to the buffer nodes in N_2 is exactly 2. This implies that $|V'_{big}| - |N_1| \leq |N_2|$. Thus, $|V'_{big}| \leq |N_1| + |N_2| = |N_{G'}(V'_{big})|$.

Since $|V'_{big}| \leq |N_{G'}(V'_{big})|$, by Hall's Theorem, a matching $\mathscr{M}$ covering V_{big} exists. Hence, algorithm BigSmall_$\ell_s\ell_b$ computes an orientation if $OPT \leq 1$ and reports `FAIL` otherwise.

Consider the orientation produced by the algorithm. If vertex v has $\beta_v = k$, then the edge from v to t^* in N has capacity zero which implies that no edge is oriented towards v, so the load of v is $q_v \leq 1$. Next we check when v has $\beta_v < k$.

- First, let v be a vertex with a big edge oriented towards it. Since a big edge oriented towards v implies a big edge node sends at least $\lceil k/2 \rceil$ units of flow to the vertex node for v, at most

$(k - \beta_v) - \lceil k/2 \rceil$ units of flow can be additionally sent to this vertex node by small edge nodes; hence at most $(k - \beta_v) - \lceil k/2 \rceil$ additional small edges are oriented towards v. The load of vertex v is at most $\ell_b + \ell_s((k - \beta_v) - \lceil k/2 \rceil) + \beta_v \ell_s$.

- Second, if vertex v has no big edge oriented towards it, since the capacity from any vertex node v to t^* is $k - \beta_v$, at most $k - \beta_v$ small edges are oriented towards it. Thus, the load of v is at most $\ell_s(k - \beta_v) + \beta_v \ell_s$

Hence, the load of a vertex v is at most

$$\begin{aligned}
&\beta_v \ell_s + max\{\ell_s(k - \beta_v), \ell_b + \ell_s((k - \beta_v) - \lceil k/2 \rceil)\} \\
&\leq max\{\ell_s k, \ell_b + \ell_s(k - k/2)\} \\
&\leq max\{k/k, \ell_b + k/(2k)\}, \text{ as } k = \lfloor 1/\ell_s \rfloor \text{ so } \ell_s \leq 1/k \\
&\leq 1 + 1/2 = 3/2. \quad \blacksquare
\end{aligned}$$

3.1.3 Step 5.2

Here we cover the case when $1/(k+1) < \ell_s \leq 1/k$ and $(k-1)/k \leq \ell_b < k/(k+1)$. Note that it is possible that $\ell_s + \ell_b \leq 1$. If $OPT \leq 1$, at most one big edge can be oriented along with one small edge toward the same vertex, as

$$2\ell_s + \ell_b > 2\left(\frac{1}{k+1}\right) + \frac{k-1}{k} = \frac{2}{k+1} - \frac{1}{k} + 1 = \frac{k-1}{k(k+1)} + 1 \geq 1, \text{ for all } k \geq 1;$$

we exploit this property below.

Lemma 3.1.3 *For any integer $k \geq 2$, there is a polynomial-time algorithm for the graph balancing problem with two rational lengths ℓ_s, ℓ_b, where $1/(k+1) < \ell_s \leq 1/k$ and $(k-1)/k \leq \ell_b < k/(k+1)$ that finds a solution of value at most $3/2$ if $OPT \leq 1$.*

Proof We consider two cases: $\ell_s + \ell_b > 1$, and $\ell_s + \ell_b \leq 1$. Assuming $OPT \leq 1$, if $\ell_s + \ell_b > 1$ either at most one big edge is oriented towards a vertex or at most k small edges are oriented towards a vertex; apply Lemma 3.1.2 to obtain an orientation where each vertex has load at most $3/2$, if such an orientation exists.

From this point forward, assume $\ell_s + \ell_b \leq 1$. Recall that $2\ell_s + \ell_b > 1$. If $OPT \leq 1$, an optimal orientation either (i) has at most a big edge oriented along with a small edge towards the same vertex, or (ii) at most k small edges are oriented towards a vertex. Like Lemma 3.1.2, compute a value β_v for the dedicated load of each $v \in V$. When $q_v \geq \ell_s + \ell_b$, set $\beta_v = k$. Otherwise, if $q_v \geq \ell_b$ set $\beta_v = k - 1$, and if not, select β_v such that $q_v = \beta_v \ell_s$. The algorithm builds a modified version of the flow network N of Lemma 3.1.2, which we describe now. First, change the capacities on the arcs incident on the big edge nodes from k to $k - 1$. Second, for each $v \in V$, set the capacity of the arc from buffer node v to vertex node v to $k - 1$ instead of k. Leave the capacities from the vertex nodes to the sink t^* as $k - \beta_v$. We show this flow network in Figure 3.2. It is straightforward to see that this modified network maintains the same property that all the arcs leaving s^* are saturated in an integral maximum flow if $OPT \leq 1$.

We modify the algorithm from Lemma 3.1.2 as follows. We refer the reader to BigSmall_$\ell_s\ell_b$ (Algorithm 3). The modified flow network we described above is built for Step 1.

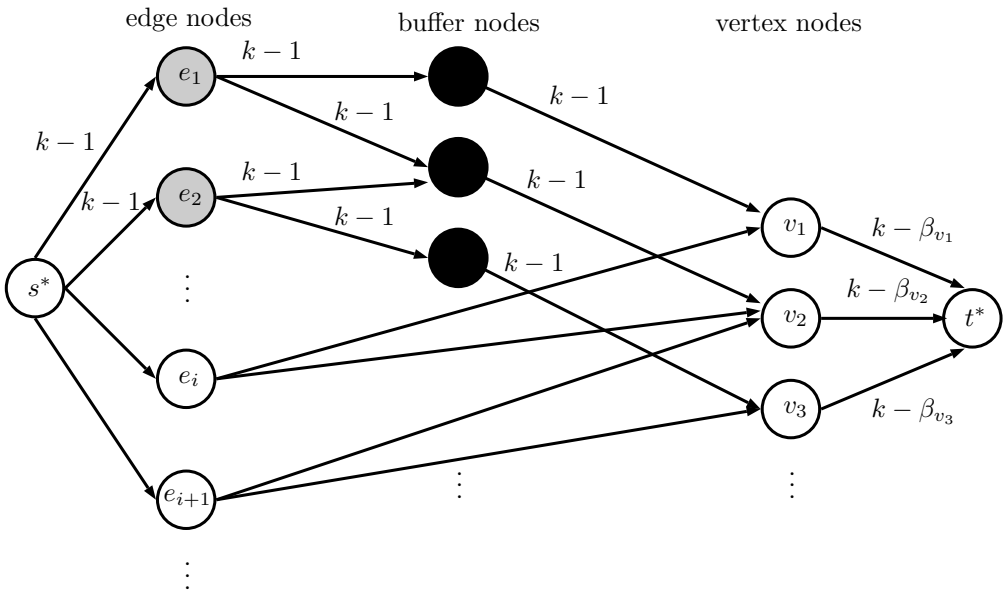

Figure 3.2: The modified flow network constructed for the case when $1/(k+1) < \ell_s \leq 1/k$ and $(k-1)/k \leq \ell_s < k/(k+1)$. Shaded nodes represent big edge nodes, and every black node is a buffer node associated with a vertex node. Assume arcs that are unlabelled have flow capacity 1.

Steps 2–3 are the same as before. In Step 4, when constructing the bipartite graph G' between the big edge nodes and buffer nodes, V_{big} remains the same, but V_{rec} contains buffer nodes that receive at least $\lceil (k-1)/2 \rceil$ units of flow from a big edge node and

$$E' = \{(u,v) \mid u \in V_{big}, v \in V_{rec}, f(u,v) \geq \lceil (k-1)/2 \rceil\}.$$

Steps 5–8 remain the same as before. Since the capacities of the arcs leaving the buffer nodes and the incoming flow to the big edge nodes are one less than in the network in Lemma 3.1.2, one can show a matching on G' exists if $OPT \leq 1$ by replacing k with $k-1$ and switching the even and odd cases of our original argument in Lemma 3.1.2.

Consider the load of a vertex v in the orientation produced by the algorithm. Clearly any vertex v with $q_v \geq \ell_s + \ell_b$ has $\beta_v = k$ and $k - \beta_v = 0$. No flow is sent to these vertex nodes, so the load of these vertices is at most 1. Now, examine vertices with $0 \leq q_v < \ell_s + \ell_b$. If $q_v \geq \ell_b$, then at most one small edge can be oriented towards v. Hence, the load of v when $q_v \geq \ell_b$ is at most $q_v + \ell_s \leq 1 + \ell_s \leq 3/2$ since $\ell_s \leq 1/2$. Finally, consider when $q_v = \beta_v \ell_s < \ell_b$:

- Let v be a vertex with a big edge oriented towards it. At least $\lceil (k-1)/2 \rceil$ units of flow are sent from its big edge node to v. Then, at most $(k - \beta_v) - \lceil (k-1)/2 \rceil$ small edges can be oriented along with the big edge towards v.

- Let v not have a big edge oriented towards it. At most $k - \beta_v$ small edges are oriented towards v.

Therefore, the load of v is at most

$$\begin{aligned}
&\beta_v \ell_s + max\{\ell_s(k-\beta_v), \ell_b + \ell_s((k-\beta_v) - \lceil (k-1)/2 \rceil)\} \\
&\leq max\{\ell_s k, \ell_b + \ell_s(k-(k-1)/2)\} \\
&< max\{k/k, k/(k+1) + (k+1)/2k\} \\
&= k/(k+1) + 1/(2k) + 1/2 \\
&< (k+1)/(k+1) + 1/2 = 3/2. \quad \blacksquare
\end{aligned}$$

3.1.4 Step 5.3

For the final case in Step 5 of algorithm GB2W, we apply a variant of the algorithm by Ebenlendr *et al.* (Chapter 2.2.3).

Lemma 3.1.4 *For every positive integer $k \geq 2$, there is a polynomial-time algorithm for the graph balancing problem with two rational lengths ℓ_s, ℓ_b, where $1/(k+1) < \ell_s \leq 1/k$ and $1/2 < \ell_b < (k-1)/k$ that finds a solution of value at most $3/2$ if $OPT \leq 1$.*

Proof We make the following modifications to the algorithm in Chapter 2.2.3. Consider any tree $T \subseteq T_B$. Every big edge e has length $p_e = \ell_b$, so we simplify the tree constraint of LP3 to

$$\sum_{(v,e)\in L(T)} \ell_b x_{ev} \geq \left(\sum_{(v,e)\in L(T)} \ell_b \right) - \ell_b \Leftrightarrow \sum_{(v,e)\in L(T)} x_{ev} \geq |L(T)| - 1. \tag{3.1}$$

Also, in the rounding procedure, if there is a leaf pair (v, e) where $e = \{u, v\}$, we change the leaf assignment and tree assignment in the following ways:

- *Leaf assignment*: if $p_e x_{eu} \leq 1/2$, e is oriented towards v.
- *Tree assignment*: if $p_e x_{eu} > 1/2$, then e is a big edge and the connected component of G_x^B containing e is a tree $T \in T_B$. Orient all edges in T away from v.

Graph Balancing Linear Program for Two Job Lengths

$$\begin{aligned}
x_{eu} + x_{ev} &= 1, && \text{for all } e = \{u, v\} \in E \quad (\text{Edge } e) \\
q_v + \sum_{e|v\in e} p_e x_{ev} &\leq 1, && \text{for all } v \in V \quad (\text{Load } v) \\
\sum_{(v,e)\in L(T)} x_{ev} &\geq |L(T)| - 1, && \text{for all } T \in T_B \text{ in } G \quad (\text{Tree } T) \\
x_{ev} &\geq 0, && \text{for all } e \in E \text{ and } v \in e,
\end{aligned}$$

We use the algorithm with the above modifications, and compute a feasible solution to the above LP; this LP is a special case of a LP given in Chapter 4 called LP5, where feasible solutions can be computed in polynomial time as stated in Lemma 4.2.2. If no fractional solution is found then no orientation exists; report `FAIL` if this is the case. Since modifying the

above thresholds of the leaf and tree assignments from 3/4 to 1/2 will allow all fractionally oriented edges to be rounded, the algorithm still finds an orientation in polynomial time.

We can extend the arguments by Ebenlendr *et al.* [20] to show that the modified algorithm produces orientations with makespan at most 3/2. The following conditions are maintained by each vertex $v \in V$ before and after each step in the rounding procedure.

(1) The load of v is at most 3/2.

(2) If $e \in G_x$ is incident on v, then v has load at most $1 + (\ell_b - 1/2)$.

(3) If e_B is a big edge in G_x^B incident on v, the load of v is at most 1.

(4) For any tree T that is a subgraph of G_x^B, the tree constraint (Tree T) is never violated.

It is not hard to modify the proof of Theorem 1 in [20] to prove that the above conditions. For completeness we sketch the proof. At the beginning of the algorithm, after a feasible solution for the LP is obtained, all the conditions above are satisfied and the load of each vertex is at most 1. Next, we show Conditions (1)–(4) are preserved for any vertex that changes its load during the rounding procedure.

- *Tree assignment*: In a tree assignment, only vertices in the tree $T \subseteq G_x^B$ containing big edge $e = \{u, v\}$ for which $p_e x_{eu} > 1/2$ and v is a leaf of T have their loads modified. Every vertex in T is incident with a big edge in G_x. Hence before this step is performed, by Condition (3), each one of these vertices has load at most 1. Consider vertex u' in T after the tree assignment has been performed. If $u' = v$, the load of v is decreased as a big edge is oriented away from v. If $u' \neq v$, there exists a path P in T from u' to v. Say this path begins at edge e'. As P is a subtree of T, it must satisfy our tree constraint (3.1) and so
$$x_{ev} + x_{e'u'} \geq |L(P)| - 1 = 2 - 1 = 1 \Leftrightarrow x_{e'u'} \geq (1 - x_{ev}) = x_{eu}.$$
All the edges in P are big, so $\ell_b x_{e'u'} \geq \ell_b x_{eu} > 1/2$. Hence, the load of u' increases by at most $\ell_b - \ell_b x_{e'u'} < \ell_b - 1/2$, and Conditions (1) and (2) are satisfied for vertex u'. Note that since fractional edge assignments in T have been eliminated, u' cannot be incident on a big edge following a tree assignment. Thus, Condition (3) does not apply to this case and Condition (4) is satisfied.

- *Leaf assignment*: In a leaf assignment, edges are oriented towards a leaf vertex. Say vertex v is a leaf. We consider two cases for edge $e = \{u, v\}$ such that $p_e x_{eu} \leq 1/2$: $p_e > 1/2$, and $p_e \leq 1/2$.

  If $p_e > 1/2$, then v is incident on a big edge and so by Condition (3), the load of v is at most 1 before the leaf assignment. Since $p_e - p_e x_{ev} = p_e x_{eu} \leq 1/2$, then following the leaf assignment the load of v is at most $1 + 1/2 = 3/2$.

  If $p_e \leq 1/2$, then $p_e = \ell_s$. By Condition (2), before the leaf assignment the load of v is at most $1 + (\ell_b - 1/2)$. So after the leaf assignment the load of v is at most
$$1 + (\ell_b - 1/2) + \ell_s \leq 1 + (k-1)/k - 1/2 + 1/k = 3/2.$$
  In any case, after a leaf assignment, v becomes isolated in G_x and the load of v cannot be increased any further, so Conditions (1)–(4) are satisfied.

- *Rotation*: The rotation step does not change the vertex loads so Conditions (1)–(3) hold. The argument showing that Condition (4) holds is the same as the argument given in the proof of Lemma 4.2.4 in Chapter 4, so for the sake of brevity we omit it here.

Therefore, by Condition (1), the load of a vertex is at most $3/2$. ∎

3.1.5 Approximation Ratio

Finally we prove algorithm GB2W is a $3/2$-relaxed decision algorithm for the graph balancing problem with two job lengths.

Lemma 3.1.5 *GB2W is a $3/2$-relaxed decision algorithm for the graph balancing problem with two job lengths.*

Proof Recall that in Step 1, algorithm GB2W scales the edge lengths and vertex dedicated loads by τ, then sets $\tau = 1$. Step 3 of algorithm GB2W invokes the algorithm of Lenstra *et al.* (Chapter 2.2.1) if $\ell_s, \ell_b \in (0, 1/2]$. By Theorem 2.2.1, either an orientation with makespan $1 + max\{\ell_s, \ell_b\} \leq 1 + 1/2 = 3/2$ is found, or $OPT > 1$ and an orientation is not produced. For Steps 4 and 5 of algorithm GB2W, Lemmas 3.1.1–3.1.4 ensure that either a solution of value at most $3/2$ is found if $OPT \leq 1$. Therefore, if $OPT \leq 1$, Step 6 of the algorithm GB2W returns an orientation with maximal load $3/2$; otherwise it returns `FAIL`. ∎

As GB2W is a $3/2$-relaxed decision algorithm, it follows that we can use the binary search procedure described in Chapter 1.1.4. This binary search is guaranteed to find a value $\tau \leq OPT$ and an orientation with load at most $3\tau/2$. For a weighted multigraph $G = (V, E, \mathbf{p}, \mathbf{q})$, since $OPT \leq |E|\ell_b$, the number of iterations of the binary search is at most $O(\log |E| + \log \ell_b)$ and since algorithm GB2W has polynomial running time, the overall running time is also polynomial.

Theorem 3.1.6 *There is a $3/2$-approximation algorithm for the graph balancing problem with two job lengths.*

3.2 A Simpler $3/2$-Approximation Algorithm

Two drawbacks of the algorithm presented in the previous section are that it is somewhat complicated and it requires using linear programming. The algorithm we present here is combinatorial, so it does not require any linear programming, and is simpler. The algorithm we give utilizes a combination of two approaches: a rounding procedure by Ebenlendr *et al.* [20] for the graph balancing problem, and a simple flow-based approach based on the flow network construction of Kolliopoulos and Moysoglou [66].

We present a $3/2$-relaxed decision algorithm Simpler_GB2W (Algorithm 4) below. To simplify our arguments, in Step 1 we rescale the edge lengths by $1/\ell_s$ so that we let the edge lengths be $k = \ell_b/\ell_s$ and $\ell_s/\ell_s = 1$; we define an edge as a *k-edge* if it has length k, and as a *1-edge* otherwise. This shrinks the length of the schedule/orientation uniformly by a factor of $1/\ell_s$. So, without loss of generality, let the edges have lengths 1 and k respectively, where $k > 1$ need not be integer. Thus, when we consider τ below, τ need not be integer.

One assumption we will make in this section is that dedicated loads are self-loops in G; these self-loops have length either 1 or k. Clearly there is no feasible schedule when $k > \tau$, so in Step 2 the algorithm returns `FAIL` if $k > \tau$. In addition, if any case below fails to produce an orientation of the edges, the algorithm returns `FAIL`.

Algorithm 4: Simpler_GB2W ($G = (V, E, \mathbf{p}, \mathbf{q}), \tau$)

Input: Multigraph G, value τ.

Output: An orientation γ for the edges in E with makespan at most $3\tau/2$, or `FAIL` if there is no orientation with makespan at most τ.

1. Divide ℓ_s, ℓ_b, and edge lengths by ℓ_s so lengths are 1 and $k := \ell_b/\ell_s$; set $\tau := \tau/\ell_s$.
2. **If** there is an edge $e \in E$ with length $p_e > \tau$ **then return** `FAIL`
3. **If** $\tau \geq 2k$ **then** apply the algorithm given in Lemma 3.2.1.
4. **If** $\tau < 2k$ **then**:
   - 4.1. **If** τ is not an integer **then** apply the algorithm in Lemma 3.2.2.
   - 4.2. **Else** apply the algorithm in Lemma 3.2.3.
5. **If** any of the algorithms used in Steps 3–4 do not compute a solution **then return** `FAIL`; otherwise **return** the orientation computed in the above steps.

Lemma 3.2.1 *Let k be a rational number, where $1 < k \leq \tau/2$. There is a polynomial-time algorithm for the graph balancing problem with two lengths $1, k$, that finds a solution of value at most $3\tau/2$ if $OPT \leq \tau$.*

Proof Note that every edge has length at most $k \leq \tau/2$. First, produce a fractional solution in the following way. Construct a bipartite flow network N', where one side has one edge node for each edge of the multigraph $G = (V, E, \mathbf{p}, \mathbf{q})$ and the other side has a vertex node for each vertex of G as shown in Figure 3.3. There is an arc from each edge node e to a vertex node v if e is incident with v in G, and the capacity of arc (e, v) is the length p_e of edge $e \in E$; each arc (s^*, e) leaving the source s^* has capacity p_e, and every arc of the form (v, t^*) where t^* is the sink has capacity τ.

Now we show that if $OPT \leq \tau$, then there is a maximum flow f that saturates all the arcs leaving the source s^*. Consider an orientation γ^* with makespan at most τ. We build a flow function f based on this orientation: for each edge $e \in E$ oriented toward vertex v, send flow of value p_e from s^* to edge node e, then send p_e units of flow from e to vertex node v. Now, we must show that the total flow received at a vertex node v can be sent to sink t^*. The capacity $c(v, t^*) = \tau$ for each vertex node v, and

$$\sum_{f(e,v)>0} f(e, v) = \sum_{e|\gamma^*(e)=v} p_e \leq \tau,$$

thus the flow sent to v can be sent to t^*. Therefore, if $OPT \leq \tau$ all the arcs leaving s^* are saturated in f; no additional flow can leave s^*, so f is a maximum flow.

A fractional solution $\mathbf{x}$ has a variable $x_{ev} \in [0, 1]$ for each $e \in E$ and endpoint v of e; $x_{ev} = 1$ when edge e is directed toward vertex v. Compute a maximum flow f on flow network N'.

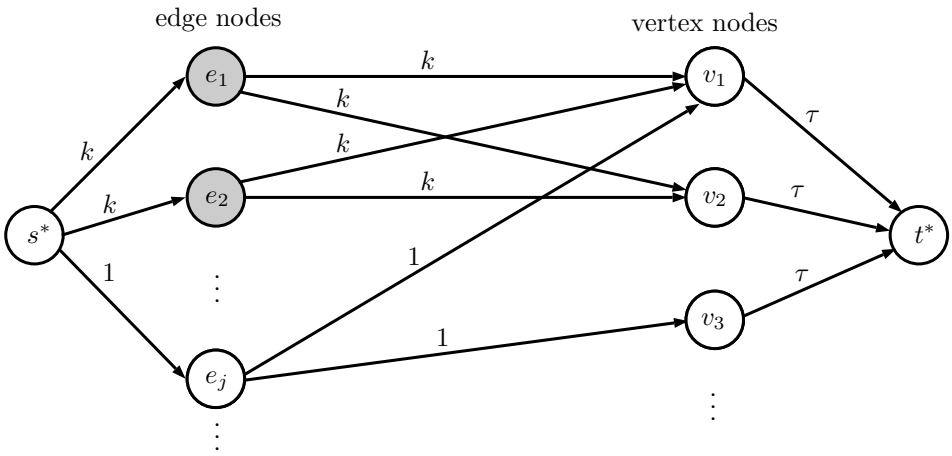

Figure 3.3: A bipartite flow network N' used to compute a fractional solution. The shaded nodes correspond to edges of length k.

Set $x_{ev} = f(e,v)/p_e$ where p_e is the length of e. For every edge $e = \{u,v\} \in E$, $f(s^*,e) = p_e$, and so $x_{eu} = f(e,u)/p_e$ and $x_{ev} = f(e,v)/p_e$. Note that all the edges are fractionally oriented as

$$x_{eu} + x_{ev} = \frac{f(e,u)}{p_e} + \frac{f(e,v)}{p_e} = \frac{f(e,u) + f(e,v)}{p_e} = 1.$$

Since the capacity $c(v,t^*) = \tau$, for all vertex nodes $v \in V$, the resulting fractional solution has makespan at most

$$\sum_{e|v \in e} p_e x_{ev} = \sum_{f(e,v)>0} f(e,v) \leq \tau.$$

If any $x_{ev} = 1$, direct e toward v. All that remains is to round the variables with $0 < x_{ev} < 1$.

While there is a cycle C in G consisting only of edges e with non-integral variables x_{ev}, apply a rotation (as described in Chapter 2.2.3) to C. As was stated in Chapter 2.2.3, rotations do not change the loads of any vertices, so $\mathbf{x}$ remains feasible. Once no more cycles exist, all the edges that correspond to non-integral variables form a forest F. For each tree in this forest, pick an arbitrary vertex as the root and orient all edges in the tree away from the root.

Prior to directing the edges in each tree, $\mathbf{x}$ was a feasible solution and the makespan was at most τ. Directing in each tree of F the edges as described above orients at most one additional edge toward each vertex. Thus the load of any vertex is no more than $\tau + \max_{e \in E} p_e = \tau + k \leq \tau + \tau/2 = 3\tau/2$. ∎

If $\tau < 2k$, then $k > \tau/2$ and so in an orientation with makespan at most τ at most one k-edge is oriented toward a vertex. We consider the two cases given by Step 4 of algorithm Simpler_GB2W, in Lemma 3.2.2 and Lemma 3.2.3. In both these cases, we build the flow network N'' of Kolliopoulos and Moysoglou [66] as shown in Figure 3.4. Flow network N'' is similar to N' from Lemma 3.2.1, except it adds a level of buffer nodes to ensure that vertex nodes do not receive flow of value more than k from k-edge nodes.

Lemma 3.2.2 *Let k be a rational number, where $1 < k \leq \tau$. There is a polynomial-time algorithm for the graph balancing problem with two lengths $1, k$, where $\tau < 2k$ and τ is not an integer, that finds a solution of value at most $3\tau/2$ if OPT $\leq \tau$.*

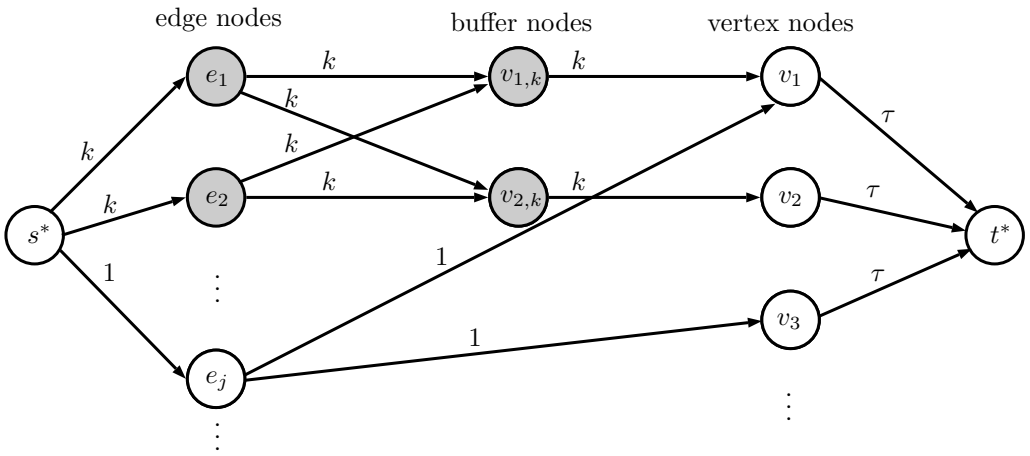

Figure 3.4: The flow network N'' of Kolliopoulos and Moysoglou [66]. The shaded edge nodes represent k-edges. Only nodes of k-edges are connected to buffer nodes. The buffer nodes limit the flow so that no more than k units of flow can be received by a vertex node from k-edges in the network.

Proof First, if τ is not an integer, then compute the largest non-negative integer p such that $0 \leq \tau - (k + p) < 1$; p exists as $\tau \geq k$. Since all edges have length either 1 or k, the load of every vertex $v \in V$ in an orientation of G must be of the form $ak + b$, where $a \in \{0, 1\}$ and b is a non-negative integer such that $0 \leq b \leq p$. Thus, if there is an orientation with makespan at most τ, there must be an orientation with makespan at most $k + p$; so we set $\tau := k + p$. A vertex that has load at most τ is either a vertex with at most one k-edge and p 1-edges oriented toward it, or has at most $k + p$ many 1-edges directed toward it.

Build flow network N'', and then compute a maximum flow f of N'' as follows.

1. Round the capacity of each arc of N'' with capacity k down to $\lfloor k \rfloor$ and then compute an integral maximum flow f. Note that the 1-edges get integral assignments, i.e. for each 1-edge e there is a vertex v such that $f(s^*, e) = 1$ and $f(e, v) = 1$. After computing the maximum flow f, set the capacity of each arc changed in N'' back to its original value.

2. Let L be the set of k-edges. Build a bipartite graph $G' = (L \cup V, E_b)$ with the k-edges L and vertices V; $e \in L$ is adjacent in G' to vertex $v \in V$ if e is incident with v in G. Compute a maximum matching $\mathscr{M}$ that pairs each k-edge $e \in L$ with a unique vertex $\mathscr{M}(e)$. If $OPT \leq \tau$, there is an orientation where each k-edge $e \in L$ is oriented towards a unique vertex $v \in V$ and since there is an edge in the bipartite graph G' with endpoints e and v for each $e \in L$, then $\mathscr{M}$ must exist.

3. Let $v_{\mathscr{M}(e),k}$ be the buffer node for vertex node $\mathscr{M}(e)$. Send additional flow f' of value $k - \lfloor k \rfloor$ along each path s^*, e, $v_{\mathscr{M}(e),k}$, $\mathscr{M}(e)$, t^* for all $e \in L$. Note that the arcs in the above path have enough residual capacity to carry this flow as f is integral and the first three arcs have capacity k; the last arc has capacity τ, and since $\tau = k + p$ for some non-negative integer p and f is integral, the last arc can carry this additional flow to t^*.

4. Set $f := f + f'$. If $OPT \leq \tau$, each arc (s^*, e) is saturated in f, so no additional flow can be sent from the source s^* and f is a maximum flow.

Now we use the maximum flow f to construct an orientation. Orient each 1-edge e toward vertex v if 1 unit of flow is sent from e to v in flow f. For each k-edge e, direct e toward the vertex v if the corresponding k-edge node sends flow of value more than $k/2$ to v.

If some k-edges send $k/2$ units of flow to two vertices, build a bipartite graph G'' with edge nodes corresponding to the k-edges sending exactly $k/2$ units of flow to two vertices on one side and vertex nodes receiving the flow from these k-edges on the other side; there is an edge from a k-edge node e to a vertex node v if e sends $k/2$ units of flow to v. Since the capacity of every arc leaving the buffer nodes is k, every node in this bipartite graph must have degree at most two, implying this bipartite graph consists only of disjoint paths and cycles. To orient each k-edge e toward a vertex v, we proceed as follows: If there is a cycle C, assign a direction along C to all the edges in C so that each node in C has at most one edge directed toward it, for each directed edge (e, v) in C, in G direct edge e toward v (see Figure 3.5); if there is a path P in G'', select a node u' with degree 1 in P and assign a direction to all the edges in the path away from this node. If u' is a vertex node, then for each (v, e) in P, in G direct edge e toward v; otherwise, u' is an k-edge node and orient in G edge e toward v if (e, v) is a directed edge in P.

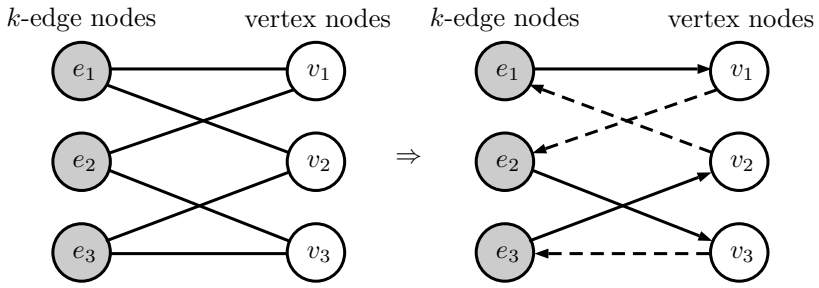

Figure 3.5: Example bipartite graph of k-edges and vertices when k-edges send exactly $k/2$ units of flow to two vertex nodes in f (left), and the assignment of k-edge nodes to vertex nodes given with solid lines (right). In this example bipartite graph, the assignment orients e_1 toward v_1, e_3 toward v_2, and e_2 toward v_3.

It follows from the way the flow f was modified that a vertex v for which the above process on bipartite graph G'' does not orient a k-edge toward it has load at most $k + p = \tau$; so we consider the load of a vertex v for which this process orients a k-edge e toward it.

After Step 4 of the algorithm, $f(s^*, e) = k$ for every k-edge node e in N'', i.e. every arc (s^*, e) is saturated. In addition, every k-edge in G has at most two endpoints, this corresponds to at most two outgoing arcs from each k-edge node in N''. So, by flow conservation, there must be a vertex node that receives at least $k/2$ units of flow from a k-edge node. If k-edge e is oriented toward v in G, then either its corresponding k-edge node e sends flow of value more than $k/2$ to vertex node v, or k-edge node e sends exactly $k/2$ units of flow to two vertices, one of which is vertex node v. Regardless, as vertex node v receives at least flow of value $k/2$ from the k-edge node e, at most $k - k/2 = k/2$ additional flow (corresponding to 1-edges being directed toward v) can be sent to v in N''. By directing k-edge e toward vertex v in G, the load of v increases by at most $k/2 \leq \tau/2$. Prior to orienting e, the load of v is at most τ. Therefore, the load of a vertex is at most $\tau + \tau/2 = 3\tau/2$. ∎

Lemma 3.2.3 *Let k be a rational number, where $1 < k \leq \tau$. There is a polynomial-time algorithm for the graph balancing problem with two lengths $1, k$, where $\tau < 2k$ and τ is an integer, that finds a solution of value at most $3\tau/2$ if $OPT \leq \tau$.*

Proof Here we use the same arguments present in the proof of Lemma 3.2.2. If τ is an integer but k is rational, then an orientation that has makespan τ must have either some vertex with τ 1-edges directed toward it or both a k-edge and $\tau - \lceil k \rceil$ 1-edges directed toward it. Furthermore, any vertex that has a k-edge oriented toward it must have load smaller than τ. Increase the length of each k-edge to $\lceil k \rceil$. Observe since $\lceil k \rceil \leq \tau$, the makespan is still at most τ.

In the flow network N'' described in the proof of Lemma 3.2.2 (see Figure 3.4), set the capacity of every arc with capacity k to $\lceil k \rceil$, and compute an integral maximum flow f. If $OPT \leq \tau$, this flow saturates all arcs (s^*, e); this can be shown using an argument similar to the one given for network N' in the proof of Lemma 3.2.1.

Orient the edges of G using f as described in the proof of Lemma 3.2.2; with the k-edges, replace $k/2$ with $\lceil k \rceil/2$ when considering the value of flow received by a vertex node from a k-edge node in N''. As each arc (s^*, e) is saturated, $\lceil k \rceil$ units of flow are sent to each k-edge node. Every edge in G can be oriented toward one of at most two vertices, so there are at most two outgoing arcs from each k-edge node in N''. If the algorithm orients a k-edge e toward vertex v in G, flow of value at least $\lceil k \rceil/2$ is received by vertex node v from k-edge node e. Furthermore, each vertex node receives no more than $\lceil k \rceil$ units of flow from k-edge nodes via its buffer node. So at most $\lceil k \rceil - \lceil k \rceil/2 = \lceil k \rceil/2$ additional units of a flow can be received by a vertex node v in N'' if a k-edge is oriented toward v in G by the algorithm. Since $\lceil k \rceil \leq \tau$ and each additional unit of flow can be an additional 1-edge being directed toward a vertex, the makespan of the orientation is at most $\tau + \lceil k \rceil/2 \leq 3\tau/2$. ∎

It follows from Lemmas 3.2.1–3.2.3 that algorithm Simpler_GB2W is 3/2-relaxed decision algorithm. Therefore, there is a combinatorial 3/2-approximation algorithm for the graph balancing problem with two job lengths.

Theorem 3.2.4 *There is a combinatorial 3/2-approximation algorithm for the graph balancing problem with two job lengths.*

Chapter 4

Graph Balancing Problem with Two Speeds and Two Job Lengths

Recall that in the graph balancing problem with two speeds and two job lengths every job J_j can be processed by one of a subset $\mathscr{M}_j$ of at most two machines with processing time p_j/s_i, where p_j is the length of J_j and s_i is the speed of the machine M_i that processes J_j, each job length $p_j \in \{\ell_s, \ell_b\}$ and every machine speed $s_i \in \{\mathfrak{s}_s, \mathfrak{s}_f\}$, $\ell_s < \ell_b$ and $\mathfrak{s}_s < \mathfrak{s}_f$. Like in the previous chapter, we will intepret the graph balancing problem as a weighted multigraph where the jobs are edges and the machines are vertices. That is, we are given a weighted multigraph $G = (V, E, \mathbf{p}, \mathbf{q}, \mathbf{s})$ with $\mathbf{s} = (s_{v_1}, s_{v_2}, \ldots, s_{v_{|V|}})$ being the speeds of the vertices and the goal is orient all the edges of G so as to minimize the makespan.

In this chapter we present a $(\sqrt{65}+7)/8$-approximation algorithm for the graph balancing problem with two speeds and two job lengths. As an ingredient for our algorithm, in Chapter 4.1 we give a $(2-\ell_s/\ell_b)$-approximation algorithm for the restricted assignment problem with two job lengths $p_j \in \{\ell_s, \ell_b\}$ when the parallel machines are uniform ($Q|\mathscr{M}_j, p_j \in \{\ell_s, \ell_b\}|C_{max}$); this extends a $(2-\ell_s/\ell_b)$-approximation algorithm by Chakrabarty *et al.* [10] for the restricted assignment problem with two job lengths on machines of speed 1. Then, we present our main approximation algorithm in Chapter 4.2. In Chapter 4.3 we present integrality gap results for the linear programs used for our approximation algorithm.

In both of the approximation algorithms presented in this chapter, we assume the job lengths are rescaled so that any processing time p_j/s_i for a job with length p_j on a machine with speed s_i is integral. One way to accomplish this is by multiplying each job length by the product of the speeds $s_1 s_2 \ldots s_m$. After this, the processing time of a job on a machine is integral, and thus the makespan of a schedule is also guaranteed to be integral. Unless stated, we will assume the job lengths ℓ_s and ℓ_b are their rescaled values.

Like in the previous chapter, the approximation algorithms in this chapter use ρ-relaxed decision procedures (see Chapter 1.1.4).

4.1 Approximation Algorithm for $Q|\mathscr{M}_j, p_j \in \{\ell_s, \ell_b\}|C_{max}$

For makespan estimate τ, our algorithm builds a single-source single-sink flow network N that is comprised of *job vertices* and *machine vertices*. For each job J_j, we include a *small* job

vertex if $p_j = \ell_s$, and a *big* job vertex if $p_j = \ell_b$. For each machine there is a *small* machine vertex and a *big* machine vertex. Add an arc with capacity 1 from the source to each job vertex. If a job J_j can be scheduled on machine M_i, we have two cases. If job vertex J_j is small, then add arcs with capacity 1 from J_j to big machine vertex M_i and small machine vertex M_i. If it is a big job vertex, then only add one arc from job vertex J_j to big machine vertex M_i with capacity 1. Finally, for each machine M_i, add an arc from big machine vertex M_i to the sink with capacity $\lfloor(\tau s_i)/\ell_b\rfloor$ and add an arc from small machine vertex M_i to the sink with capacity $\lfloor(\tau s_i)/\ell_s\rfloor - \lfloor(\tau s_i)/\ell_b\rfloor$. We show flow network N in Figure 4.1.

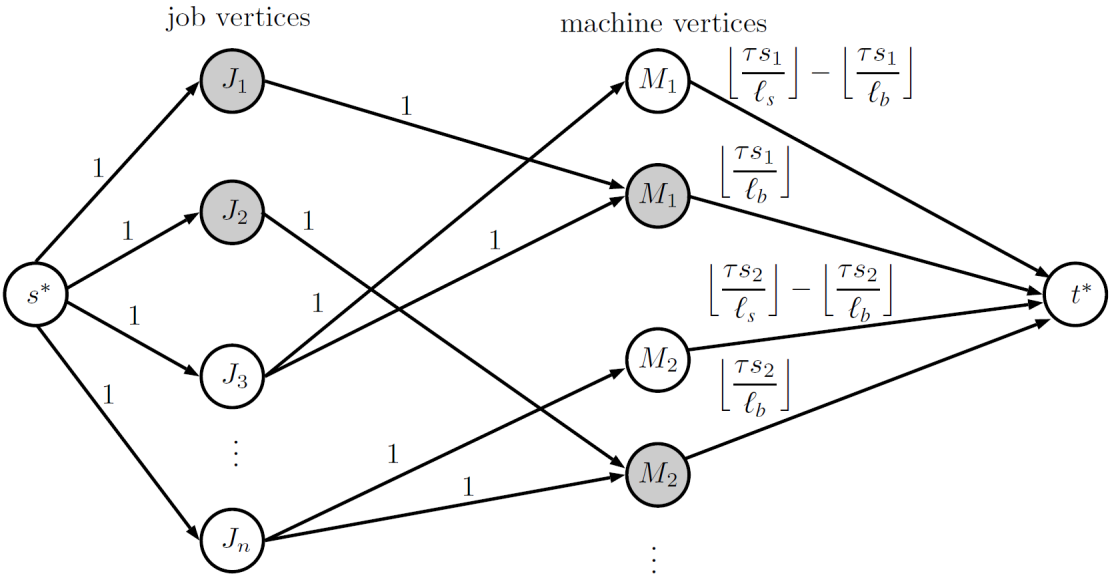

Figure 4.1: Flow network N with source s^* and sink t^*. The shaded vertices are big and the non-shaded machine and job vertices are small.

Now we describe a $(2-\ell_s/\ell_b)$-relaxed decision algorithm for $Q|p_j \in \{\ell_s, \ell_b\}, \mathscr{M}_j|C_{max}$. Build the flow network N and compute an integral maximum flow f of N. If not all the arcs leaving the source are saturated by f then by Lemma 4.1.1 below there is no schedule of makespan at most τ. Otherwise, for each job vertex J_j and machine vertex corresponding to machine M_i with flow $f(J_j, M_i) = 1$, schedule job J_j on machine M_i.

Lemma 4.1.1 *In the flow network N given above, a maximum integral flow f saturates all the arcs leaving the source if $OPT \leq \tau$.*

Proof Consider a schedule S with makespan at most τ. We construct a flow function f that saturates all the arcs leaving the source using this schedule: for each job vertex J_j in N, send 1 unit of flow from the source to J_j. If job J_j is scheduled in S on machine M_i, we have two cases:

1. $p_j = \ell_b$. Send 1 unit of flow from job vertex J_j to the big machine vertex for machine M_i and then to the sink.

2. $p_j = \ell_s$. If the arc leaving small machine vertex M_i is not saturated, then send 1 unit of flow from job vertex J_j to small machine vertex M_i and then to the sink. Otherwise, send this unit of flow to big machine vertex M_i and then to the sink.

Now we show that the capacity constraints are not violated. Let n_{i,ℓ_s} and n_{i,ℓ_b} be the number of jobs of lengths $p_j = \ell_s$ and $p_j = \ell_b$ scheduled on machine M_i in S, respectively. If the makespan of S is τ, then

$$n_{i,\ell_s} + n_{i,\ell_b} \leq \left\lfloor \frac{\tau s_i}{\ell_b} \right\rfloor + \left(\left\lfloor \frac{\tau s_i}{\ell_s} \right\rfloor - \left\lfloor \frac{\tau s_i}{\ell_b} \right\rfloor \right) = \left\lfloor \frac{\tau s_i}{\ell_s} \right\rfloor. \tag{4.1}$$

The flow capacity of the arc leaving big machine vertex M_i to the sink is $\lfloor (\tau s_i)/\ell_b \rfloor$, and since $n_{i,\ell_b} \leq \lfloor (\tau s_i)/\ell_b \rfloor$, the flow from the big job vertices can be sent through the big machine vertices to the sink. By (4.1), the sum of the capacities of the arcs from big and small machine vertices corresponding to machine M_i to the sink minus the flow sent from big job vertices is

$$\left(\left\lfloor \frac{\tau s_i}{\ell_b} \right\rfloor - n_{i,\ell_b} \right) + \left(\left\lfloor \frac{\tau s_i}{\ell_s} \right\rfloor - \left\lfloor \frac{\tau s_i}{\ell_b} \right\rfloor \right) = \left\lfloor \frac{\tau s_i}{\ell_s} \right\rfloor - n_{i,\ell_b} \geq n_{i,\ell_s}.$$

Hence, the flow can be sent from the small job vertices through the machine vertices to the sink. Since all the flow sent from the source must pass through the job vertices, no additional units of flow can be sent through the flow network N. Therefore f is a valid maximum integral flow that saturates all the arcs leaving the source. ∎

Lemma 4.1.2 *Our algorithm produces a schedule with makespan at most* $(2-\ell_s/\ell_b)\tau$ *if* $OPT \leq \tau$.

Proof Consider a machine M_L that finishes last in the schedule. The load of machine M_L is at most

$$\frac{\ell_b}{s_L}\left\lfloor \frac{\tau s_L}{\ell_b} \right\rfloor + \frac{\ell_s}{s_L}\left(\left\lfloor \frac{\tau s_L}{\ell_s} \right\rfloor - \left\lfloor \frac{\tau s_L}{\ell_b} \right\rfloor \right) = \left(\frac{\ell_b - \ell_s}{s_L} \right) \left\lfloor \frac{\tau s_L}{\ell_b} \right\rfloor + \frac{\ell_s}{s_L}\left\lfloor \frac{\tau s_L}{\ell_s} \right\rfloor$$

$$\leq \left(1 - \frac{\ell_s}{\ell_b}\right)\tau + \tau = \left(2 - \frac{\ell_s}{\ell_b}\right)\tau. \quad \blacksquare$$

Theorem 4.1.3 *Let* $\ell_s, \ell_b \in \mathbb{Z}^+$, *where* $\ell_s < \ell_b$. *There is a* $(2 - \ell_s/\ell_b)$*-approximation algorithm for* $Q|\mathcal{M}_j, p_j \in \{\ell_s, \ell_b\}|C_{max}$.

4.2 Algorithm for the Graph Balancing Problem with 2 Speeds and 2 lengths

Let $\Delta = (\sqrt{65} - 1)/8$. For convenience we scale the speeds so that $\vartheta_s = 1$ and $\vartheta_f > 1$, $\vartheta_f \in \mathbb{Q}^+$. Our approximation algorithm uses the $(1+\Delta)$-relaxed decision algorithm described below. Note that in Step 1 all the edge lengths and dedicated loads are divided by τ so that the algorithm looks for a solution with makespan at most $1 + \Delta$.

Algorithm 5: GB2S2W($G = (V, E, \mathbf{p}, \mathbf{q}, \mathbf{s}), \tau$)

Input : Multigraph $G = (V, E, \mathbf{p}, \mathbf{q}, \mathbf{s})$, value τ.
Output: An orientation for the edges in E with makespan at most $(1 + \Delta)\tau$, or reports `FAIL` if there is no orientation with makespan at most τ.

1. Divide edge lengths and vertex dedicated loads by τ; set $\tau = 1$.
2. Refine G and apply preliminary processing, as given in Chapter 4.2.1.
3. Construct multigraph G' as described in Chapter 4.2.2.
4. Solve linear program LP6 as described in Chapter 4.2.3 to obtain fractional solution $\mathbf{x}$; **if** no solution is found **then** report `FAIL` and go to Step 6.
5. Round $\mathbf{x}$ using the rounding procedure given in Chapter 4.2.4.
6. **If** any of the algorithms used in Steps 2–5 reports `FAIL` **then return** `FAIL`; otherwise **return** the orientation found.

4.2.1 Refining G and Preliminary Processing

For each edge e of G, direct e towards one of its endpoints if orienting it towards the other endpoint exceeds makespan 1. As each edge e is directed toward some vertex v, it is removed from G and its length is added to the dedicated load of v. If G contains no edges and no vertex load is larger than 1, then an orientation with makespan at most 1 is found. Report `FAIL` if any vertex has load greater than 1; otherwise, $q_v/s_v \leq 1$ for every $v \in V$, and $p_e/s_v \leq 1$ for any $v \in V$ and $v \in e$.

Now, depending on the values of ℓ_s and ℓ_b, we may perform one of the steps below.

Lemma 4.2.1 *There is a* $(1 + \Delta)$*-approximation algorithm for the graph balancing problem with two speeds and two lengths when either:*

1. $\ell_s > 1$,
2. $2 - \ell_s/\ell_b \leq 1 + \Delta$,
3. $\ell_s/\vartheta_f > 1/2$, *or*
4. *all edges* e *with* $v \in e$ *satisfy* $p_e/s_v \leq \Delta$.

Proof Below we consider each situation listed above by its respective number.

1. If $\ell_s > 1$, then every edge must be incident to a vertex with speed ϑ_f, so this is just an instance of the graph balancing problem with two lengths. Set the length of every edge to p_e/ϑ_f and each dedicated load to q_v/ϑ_f. Use one of the 3/2-relaxed decision algorithms in Chapter 3 using estimate of the makespan τ to try to compute an orientation with makespan at most 3/2. If there is no such orientation, report `FAIL`.

2. If $2 - \ell_s/\ell_b \leq 1 + \Delta$, we use the algorithm of Chapter 4.1 with estimate of the makespan τ to compute an orientation.

3. If $\ell_s/\mathfrak{s}_f > 1/2$, then $p_e/s_v > 1/2$ for every $e \in E$ and $v \in e$. Hence, if $OPT \leq 1$, at most one edge can be oriented towards a vertex and so G must be a pseudoforest. So we can employ a generalization of Algorithm 2 from Chapter 3. We can find an orientation with makespan at most 1 or determine that $OPT > 1$ as follows:

   (a) For each cycle $\mathscr{C} \subseteq G$ mark each vertex in $\mathscr{C}$, then select from the two possible orientations for its edges the one with minimum makespan. Remove the edges of $\mathscr{C}$ from G.

   (b) Now there are no more cycles in G. Consider each tree $T \subseteq G$; we have two possibilities:

      (i) If T contains one marked vertex, orient the edges in T away from that vertex.

      (ii) Otherwise, choose in turn each vertex as the root and orient the edges in T away from it. Select the orientation with minimum makespan.

   If the selected orientation has makespan larger than 1, report `FAIL`.

4. If every edge e in G with $v \in e$ satisfies $p_e/s_v \leq \Delta$, apply Theorem 2.2.1 to obtain an orientation with makespan at most $1 + \Delta$ or to determine that there is no orientation with makespan at most 1. In the latter case, report `FAIL`.

If an orientation is produced above, it has makespan at most $\max\{3/2, 1 + \Delta, 1\} \leq 1 + \Delta$. ∎

4.2.2 Step 4: Building the Multigraph G'

We construct a weighted multigraph $G' = (V', E', \mathbf{p}', \mathbf{q}', \mathbf{s}')$ from G as follows. First, set $V' = V$, $\mathbf{s}' = \mathbf{s}$ and $E' = \emptyset$. Next, for all $v \in V$, set $q'_v = q_v/s_v$, and for every edge $e = \{u, v\} \in E$ in G:

1. if $s_u = s_v$, then add to G' an edge $e' = e$ with length $p'_e = p_e$;

2. if $s_u \neq s_v$, then add to G' an *auxiliary node* w_{uv} and two *twin edges* $e' = \{u, w_{uv}\}$ and $e'' = \{w_{uv}, v\}$, both of length p_e. Set $q'_{w_{uv}} = 0$ and $s'_{w_{uv}} = 1$.

We require that for each auxiliary node its twin edges must be oriented in the same direction: one toward the auxiliary node and one away from the auxiliary node. The auxiliary nodes allow us to treat edges which contribute different loads to its endpoints as two individual edges.

4.2.3 Step 5: Linear Programming Formulation

Consider an edge $e = \{u, v\}$ in G when $s_u = s_v$. Edge e is a *big* edge in G' if $p_e/s_u > 1/2$, and is a *small* edge otherwise. Now consider when $e = \{u, v\}$ and u and v have speeds $s_u = 1$ and $s_v = \mathfrak{s}_f$ in G, respectively; in G', the edge e corresponds to an auxiliary node w_{uv} with twin edges $e' = \{u, w_{uv}\}$ and $e'' = \{w_{uv}, v\}$. We say e' is *big* if $p_e > 1/2$, and is *small* otherwise. In addition, e'' is *big* if $p_e/\mathfrak{s}_f > 1/2$, and *small* otherwise. A tree $T \subseteq G'$ is *big* if all of its edges

are big. Let T_B be the set of big trees of G'. For any subgraph $T \subseteq G'$, define the set of *leaf pairs* of T to be

$$L(T) = \{(v, e) \in V' \times E' |\ v \text{ has degree 1 in } T, v \in e, \text{ and } e \in T\}.$$

We formulate the *auxiliary graph balancing linear program 1* (LP5) shown below based on LP3 in Chapter 2.2.3. For each auxiliary node w_{uv} in G', we define four variables $x_{e'u}$, $x_{e''w_{uv}}$, $x_{e'w_{uv}}$, and $x_{e''v}$, and two constraints $x_{e'u} = x_{e''w_{uv}}$ and $x_{e'w_{uv}} = x_{e''v}$ called *auxiliary constraints*. Constraint (Load v) ensures that the load of each vertex $v \in V$ in G is no more than 1. Constraint (Edge e) guarantees that every edge is oriented. Consider any big tree $T \in T_B$ in G'. Let $\kappa_T \in \{\ell_s, \ell_b\}$ be the largest edge length in the big maximal tree T' that contains T. Every edge in T is big, so at most one edge in T can be oriented towards some vertex in G'. Furthermore, at most one big edge can be oriented away from a leaf in T. Hence, if the largest edge length in big tree T is κ_T, we obtain the following generalization of the *tree constraint* (Tree T) of Chapter 2.2.3,

$$\sum_{(v,e)\in L(T)} p_e x_{ev} \geq \sum_{(v,e)\in L(T)} p_e - \kappa_T, \text{for all } T \in T_B \text{ in } G'.$$

We will call this the tree constraint from this point forward.

Auxiliary Graph Balancing Linear Program 1 (LP5)

$$x_{eu} + x_{ev} = 1, \quad \text{for all } e = \{u, v\} \in E' \quad (\text{Edge } e)$$

$$\frac{q_v}{s_v} + \sum_{e|v\in e} \frac{p_e x_{ev}}{s_v} \leq 1, \quad \text{for all } v \in V \quad (\text{Load } v)$$

$$\sum_{(v,e)\in L(T)} p_e x_{ev} \geq \sum_{(v,e)\in L(T)} p_e - \kappa_T, \quad \text{for all } T \in T_B \text{ in } G' \quad (\text{Tree } T)$$

$$x_{e'u} = x_{e''w_{uv}}, \quad \text{for all } \{u, v\} \in E, \text{ auxiliary node}$$

$$x_{e'w_{uv}} = x_{e''v}, \quad w_{uv} \text{ and twin edges } e' \text{ and } e''(\text{Auxiliary})$$

$$x_{ev} \geq 0, \quad \text{for all } e \in E' \text{ and } v \in e,$$

Observe that LP5 can have exponentially many tree constraints. Instead of solving LP5, our algorithm solves another linear program that is very similar to LP4 called the *auxiliary graph balancing linear program 2* (LP6); LP6 has a polynomial number of constraints. Instead of using the tree constraints as in LP5, LP6 has a constraint (Star v) for each vertex $v \in V$. For each $v \in V$ in G', the *star constraint* of v states that in the star formed by the big edges incident on v, at most one big edge can be oriented towards v. We show that by solving LP6, we also obtain a feasible solution for LP5. We note that since the tree constraints of LP5 are more general than those in LP3 and LP6 is a generalization of LP4 (as speeds are introduced), our proof extends the one originally given by Ebenlendr *et al.* [20].

Auxiliary Graph Balancing Linear Program 2 (LP6)

$$
\begin{aligned}
x_{eu} + x_{ev} &= 1, && \text{for all } e = \{u, v\} \in E' && \text{(Edge } e\text{)} \\
\frac{q_v}{s_v} + \sum_{e|v\in e} \frac{p_e x_{ev}}{s_v} &\leq 1, && \text{for all } v \in V && \text{(Load } v\text{)} \\
\sum_{\substack{e|v\in e,\\ \frac{p_e}{s_v} > 1/2}} x_{ev} &\leq 1 && \text{for all } v \in V && \text{(Star } v\text{)} \\
x_{e'u} &= x_{e''w_{uv}}, && \text{for all } \{u, v\} \in E, \text{ auxiliary node} \\
x_{e'w_{uv}} &= x_{e''v}, && w_{uv} \text{ and twin edges } e' \text{ and } e''\text{(Auxiliary)} \\
x_{ev} &\geq 0, && \text{for all } e \in E' \text{ and } v \in e,
\end{aligned}
$$

Lemma 4.2.2 *Linear program LP6 has the following properties:*

1. *A feasible solution* $\mathbf{x}$ *of LP6 is also a feasible solution of LP5.*
2. *Consider the subgraph H of big edges in G′ where each big edge e satisfies* $0 < x_{ev} < 1$ *for any* $v \in e$*. H is a disjoint union of trees and simple cycles.*

Proof We prove each property separately. Consider any feasible solution $\mathbf{x}$ of LP6 for weighted multigraph G'.

1. We must show that the tree constraint (Tree T) of LP5 is satisfied for any big tree $T \in T_B$ in G'. Pick any big tree $T = (V_T, E_T) \in T_B$, and let r be the number of vertices in T. Every edge in $e \in T$ satisfies constraint (Edge e) including any twin edges incident on auxiliary nodes. At most one big edge can be oriented towards any internal vertex of T, so all the internal vertices v of T must satisfy constraint (Star v). To obtain the fractional values at the leaf pairs of T, sum $x_{eu} + x_{ev} = 1$ for each e of T from the $r - 1$ (Edge e) constraints of each $e \in E_T$, then subtract the sum of the inequalities $\sum_{\substack{e|v\in e,\\ \frac{p_e}{s_v} > 1/2}} x_{ev} = \sum_{e\in E_T | v\in e} x_{ev} \leq 1$ of the $r - |L(T)|$ (Star v) constraints over the internal vertices v of T. That is,

$$\sum_{(v,e)\in L(T)} x_{ev} = \sum_{e\in E_T, v\in e} x_{ev} - \sum_{\substack{v\in V_T|\\ deg_T(v)>1}} \sum_{\substack{e\in E_T|\\ v\in e}} x_{ev}. \tag{4.2}$$

   As each (Edge e) constraint implies $x_{eu} + x_{ev} = 1$ for each $e \in E_T$, then

$$\sum_{e\in E_T, v\in e} x_{ev} = r - 1,$$

   and since each (Star v) constraint for an internal node $v \in V_T$ implies $\sum_{\substack{e\in E_T,\\ v\in e}} x_{ev} \leq 1$,

$$\sum_{\substack{v\in V_T|\\ deg_T(v)>1}} \sum_{\substack{e\in E_T|\\ v\in e}} x_{ev} \leq r - |L(T)|.$$

Hence,

$$\sum_{e\in E_T, v\in e} x_{ev} - \sum_{\substack{v\in V_T| \\ deg_T(v)>1}} \sum_{\substack{e\in E_T| \\ v\in e}} x_{ev} \geq |L(T)| - 1,$$

and by (4.2),

$$\sum_{(v,e)\in L(T)} x_{ev} \geq |L(T)| - 1. \tag{4.3}$$

Recall that the largest edge of T has length no more than $\kappa_T > 0$, so

$$\sum_{(v,e)\in L(T)} p_e x_{ev} = \sum_{(v,e)\in L(T)} \kappa_T x_{ev} - \sum_{(v,e)\in L(T)} (\kappa_T - p_e) x_{ev}. \tag{4.4}$$

By inequality (4.3) and $x_{ev} \leq 1$ for all $e \in E_T$ and $v \in e$,

$$\begin{aligned} &\sum_{(v,e)\in L(T)} \kappa_T x_{ev} - \sum_{(v,e)\in L(T)} (\kappa_T - p_e) x_{ev} \\ &\geq \kappa_T(|L(T)| - 1) - \sum_{(v,e)\in L(T)} (\kappa_T - p_e) \\ &= \sum_{(v,e)\in L(T)} p_e - \kappa_T. \end{aligned} \tag{4.5}$$

Therefore, by (4.4) and (4.5) the tree constraint is satisfied for any big tree $T \in T_B$.

2. We now show by contradiction that H is a disjoint union of trees and simple cycles. Assume H contains a cycle C with r edges and there is an edge $e' \notin C$, $e' \in H$ incident to some vertex $v' \in C$, where the other endpoint of e' may or may not be in C. By taking the sum of each $x_{eu} + x_{ev} = 1$ of the (Edge e) constraints for all $e \in C$, we obtain

$$\sum_{e\in C} \sum_{v\in e} x_{ev} = r. \tag{4.6}$$

Next, by adding together the inequalities $\sum_{\substack{e|v\in e \\ p_e/s_v>1/2}} x_{ev} \leq 1$ of the (Star v) constraints for all the vertices v in C, we get

$$\sum_{v\in C} \sum_{\substack{e|v\in e \\ p_e/s_v>1/2}} x_{ev} \leq r. \tag{4.7}$$

Observe that the edge e' is in H and e' is incident with vertex v' in C. Then $x_{e'v'} > 0$, and by applying (4.6) and (4.7),

$$r \geq \sum_{v\in C} \sum_{\substack{e|v\in e \\ p_e/s_v>1/2}} x_{ev} \geq x_{e'v'} + \sum_{e\in C} \sum_{v\in e} x_{ev} > \sum_{e\in C} \sum_{v\in e} x_{ev} = r.$$

Therefore, by contradiction, H is a disjoint union of trees and simple cycles. ∎

As an aside, we show in Chapter 4.3 that the integrality gaps of both linear programs above are at least 5/3 for the graph balancing problem with two speeds and two lengths, and at least 7/4 for the case of three lengths and two speeds.

4.2.4 Step 6: Rounding the Fractional Solution

If LP5 does not have a solution, there is no orientation with makespan 1, and we report `FAIL`. Otherwise, we compute a solution $\mathbf{x}$ and then round it with the following rounding procedure that uses some ideas from Chapter 3 and Chapter 2.2.3. In Step 2 of our algorithm we considered certain situations for which we obtained an orientation whose makespan is at most $1 + \Delta$, these situations are given in Lemma 4.2.1. Thus we only need to round $\mathbf{x}$ for instances not covered by Lemma 4.2.1. We organize below the remaining instances into three cases and prove these are indeed the only instances left in Lemma 4.2.3. In these three cases the lengths of ℓ_s and ℓ_b are not "close" and have the following key properties: (1) all the edges of length ℓ_b incident on vertices of speed ϑ_f are small, and no big edge in G' has length larger than 1; (2) all the edges with length ℓ_b are big, and the big edges may have length larger than 1; and (3) all the edges of length ℓ_b are big and can have length larger than 1, and each edge with length ℓ_s either is big if it is incident on a vertex with speed 1 or small if it is incident on a vertex of speed ϑ_f.

Lemma 4.2.3 *After Step* 2 *of* GB2S2W*, there are only the following possible cases left:*

1. *(a)* $\ell_b/\vartheta_f \leq 1/2$, $\ell_b > 1$, $\Delta < \ell_s \leq 1$, *and* $\ell_s/\vartheta_f \leq 1/2$.
   (b) $\ell_b/\vartheta_f \leq 1/2$, $\Delta < \ell_b \leq 1$, *and* $\ell_s \leq 1/2$.
2. *(a)* $1/2 < \ell_b/\vartheta_f \leq 1$, $\ell_b > \Delta$, *and* $\ell_s \leq 1/2$.
   (b) $1/2 < \ell_b/\vartheta_f \leq 1$, $\ell_b > 1$, $1/2 < \ell_s \leq \Delta$, *and* $\ell_s/\vartheta_f \leq 1/2$.
3. $1/2 < \ell_b/\vartheta_f \leq 1$, $\ell_b > 1$, $\Delta < \ell_s \leq 1$, *and* $\ell_s/\vartheta_f \leq 1/2$.

In addition if $\ell_s > 1/2$, *then* $\Delta < (3\ell_b)/4$.

Proof We first re-organize the cases given in Lemma 4.2.3 into a new equivalent list of four cases, then we prove these new cases are the only ones left to be addressed by the algorithm. Let us first combine Cases 2(b) and 3 into Case (iii) below. Next, split Case 2(a) into two parts: when $\ell_b > 1$ we get Case (ii) below, and when $\Delta < \ell_b \leq 1$ we combine this case with Case 1(b) to get Case (i). Finally we carry over Case 1(a) as Case (iv) below. Thus, we obtain an equivalent list of cases to those in Lemma 4.2.3:

(i) $\ell_b/\vartheta_f \leq 1$, $\Delta < \ell_b \leq 1$, and $\ell_s \leq 1/2$;

(ii) $1/2 < \ell_b/\vartheta_f \leq 1$, $\ell_b > 1$, and $\ell_s \leq 1/2$;

(iii) $1/2 < \ell_b/\vartheta_f \leq 1$, $\ell_b > 1$, $1/2 < \ell_s \leq 1$, and $\ell_s/\vartheta_f \leq 1/2$;

(iv) $\ell_b/\vartheta_f \leq 1/2$, $\ell_b > 1$, $\Delta < \ell_s \leq 1$, and $\ell_s/\vartheta_f \leq 1/2$.

It is important to note that $\ell_b/\vartheta_f \leq 1$, otherwise an edge with length ℓ_b cannot be oriented without the makespan exceeding 1. Now we show the above list is complete by exhaustively considering all the possible values for ℓ_s and ℓ_b with respect to the speeds.

- $\ell_b \leq 1$. If $\ell_s > 1/2$, Case 2 of Lemma 4.2.1 applies as $2 - \ell_s/\ell_b < 2 - (1/2)/1 = 3/2 < 1 + \Delta$. Also, if $\ell_b \leq \Delta$ every $e \in E$ and $v \in e$ in G has $p_e/s_v \leq \ell_b/\vartheta_s \leq \Delta$, so Case 4 of Lemma 4.2.1 covers this situation. Thus, if $\ell_b \leq 1$ the only case that we must address is Case (i) above.

- $\ell_b > 1$. When G is modified in Step 2 of our algorithm, any edges with length ℓ_b are directed to vertices with speed ϑ_f. Case 3 of Lemma 4.2.1 covers when $\ell_s/\vartheta_f > 1/2$, and Case 1 of Lemma 4.2.1 applies when $\ell_s > 1$. If $\ell_b/\vartheta_f \leq 1/2$ and $\ell_s \leq \Delta$, Case 4 of Lemma 4.2.1 applies. Otherwise, when $\ell_b/\vartheta_f \leq 1/2$ but $\Delta < \ell_s \leq 1$, is covered by Case (iv) above.

  Finally, if $1/2 < \ell_b/\vartheta_f \leq 1$, the cases left that are not covered by Lemma 4.2.1 are when either $\ell_s \leq 1/2$ or both $1/2 < \ell_s \leq 1$ and $\ell_s/\vartheta_f \leq 1/2$; these are Case (ii) and Case (iii) above.

Finally, we prove the remark at the end of Lemma 4.2.3. If $\ell_s > 1/2$, then if $\Delta \geq (3\ell_b)/4 \Leftrightarrow \ell_b \leq (4\Delta)/3$, we apply Case 2 of Lemma 4.2.1 as

$$2 - \frac{\ell_s}{\ell_b} \leq 2 - \frac{\frac{1}{2}}{\frac{4\Delta}{3}} = 2 - \frac{3}{8\Delta} < 1 + \Delta.$$

Thus, if $\ell_s > 1/2$, then $\Delta < (3\ell_b)/4$ for any remaining cases of Lemma 4.2.3. ∎

Our rounding procedure ensures the load increase from rounding a solution of LP5 is no more than Δ by using both Lemma 4.2.3 and the tree constraint for each big tree $T \in T_B$. Unlike in the rounding given in Chapter 2.2.3, since speeds are introduced we cannot ensure that every $\kappa_T \leq 1$. As we justify later in our analysis of the makespan, when we invoke the tree constraints in our analyses, the three cases of Lemma 4.2.3 correspond to three non-trivial situations where: every $\kappa_T \leq 1$; every $\kappa_T = \ell_b$; and either $\kappa_T = \ell_s$ or $\kappa_T = \ell_b$. We introduce two *thresholds* t_{ϑ_f} and t_{ϑ_s} whose values are set according to the cases stated in Lemma 4.2.3:

1. for Case 1 we set $t_{\vartheta_f} = t_{\vartheta_s} = 5/6$;
2. for Case 2 we set $t_{\vartheta_f} = t_{\vartheta_s} = (3\ell_b)/4$; and
3. for Case 3 we set $t_{\vartheta_f} = (3\ell_b)/4$, $t_{\vartheta_s} = \Delta$.

Let $G'_{\mathbf{x}}$ be the subgraph of G' where every edge is *fractionally oriented*—i.e. every edge $e = \{u, v\} \in G'_{\mathbf{x}}$ satisfies $0 < x_{eu} < 1$. Note that $G'_{\mathbf{x}}$ has no leaves that are auxiliary nodes. It is assumed below that once an edge is oriented in $G'_{\mathbf{x}}$, it is deleted from $G'_{\mathbf{x}}$. The rounding procedure is as follows.

While $G'_{\mathbf{x}}$ has an edge, do the following.

1. If there is a leaf pair (v, e), where $e = \{v, z\}$, do the following.

   - *Leaf assignment.* If $p_e x_{ez} \leq t_{s_v}$ or e is a small non-twin edge, then orient e towards the leaf v. If e is a twin edge incident on auxiliary node $z = w_{vu}$, then direct e away from w_{vu} and direct the other twin edge e' incident on w_{vu} towards w_{vu}.
   - *Tree assignment.* If $p_e x_{ez} > t_{s_v}$, consider two cases:

(a) if e is big, then let T be the maximal big tree rooted at v containing e;

(b) if e is small, then e is a twin edge incident on some auxiliary node $z = w_{vu}$. Orient e towards w_{vu}. Let T be the maximal big tree rooted at w_{vu}.

Orient the edges in T away from its root. If any leaves of T are auxiliary nodes, then orient the small twin edges incident on them (which are not in T) away from these leaves.

2. *Rotation*. If there is no leaf pair, then find a cycle C by performing a walk starting at any vertex and appending edges to the walk where big edges are selected over small edges until a simple cycle C is found. Assign to C a direction consistent with the order in which the edges in C were appended. Next apply ROTATE($\mathbf{x}, C$) (from Chapter 2.2.3): recall that it computes $\delta \leftarrow \min_{e=(u,v)\in C}(p_e x_{eu})$ and then for each $e = (u, v) \in C$, set $x_{eu} \leftarrow x_{eu} - \delta/p_e$ and $x_{ev} \leftarrow x_{ev} + \delta/p_e$. After ROTATE, at least one edge $e = \{u, v\}$ in C has $x_{eu} = 0$ and $x_{ev} = 1$ so e is oriented to v.

Lemma 4.2.4 *After any leaf assignment or tree assignment, the auxiliary constraints and the tree constraints are satisfied. In addition, rotations preserve all the constraints of LP5.*

Proof First we prove the claim for leaf assignments and tree assignments, then we consider rotations. Assume the tree constraints and auxiliary constraints of LP5 are satisfied before a leaf assignment or tree assignment occurs.

1. *The auxiliary constraints are never violated by a leaf assignment or tree assignment.* Consider any pair of twin edges of some auxiliary node w in $G'_{\mathbf{x}}$. Either operation will orient both twin edges in the same direction and they are removed from $G'_{\mathbf{x}}$, so these auxiliary constraints are not relevant anymore.

2. *After any leaf assignment or tree assignment, for any big tree $T \in T_B$ in $G'_{\mathbf{x}}$, the constraint (Tree T) is never violated.* First consider when a leaf assignment occurs; this orients a leaf edge towards a leaf vertex. Note that if this leaf edge is a twin edge, its associated twin edge also gets directed towards the leaf. After the algorithm removes these edges, the tree constraints for any $T \in T_B$ in $G'_{\mathbf{x}}$ are still satisfied as the tree constraints were satisfied before the leaf assignment.

   Next, consider when a tree assignment occurs. Let T' be the maximal big tree built by the tree assignment. By the maximality of T', observe that the big edges incident on vertices in T' are only directed toward vertices in T' by the tree assignment. All the big edges in T' are oriented and so they are removed from $G'_{\mathbf{x}}$. Therefore, since the tree constraints were satisfied before the tree assignment, the tree constraints for any $T \in T_B$ in $G'_{\mathbf{x}}$ are not violated after the tree assignment.

Assuming the constraints of LP5 are preserved before a rotation, we show that after a rotation all the constraints of LP5 still are satisfied. A rotation begins by picking an arbitrary vertex in G', then performs a walk appending edges until a simple cycle C is found, where big edges are picked over small edges. Then, algorithm ROTATE is applied which sets $x_{eu} = x_{eu} - \delta/p_e$ and $x_{ev} = x_{ev} + \delta/p_e$ for each $e \in C$ if C is directed from u to v, where $\delta := \min_{e=(u,v)\in C}(p_e x_{eu})$.

Pick any edge $e = \{u, v\} \in E'$ in C. Since $x_{eu} + x_{ev} = (x_{eu} - \delta/p_e) + (x_{ev} + \delta/p_e)$, constraint (Edge e) is satisfied for every $e \in C$. Also, it is not hard to see that every $x_{ev} \geq 0$. Now we check the constraint (Load v) for each vertex $v \in V$ in the cycle C in G'. Pick a vertex $v \in V$ in C, and check the load contributed to v by its incident edges $e', e'' \in C$. The load on v after a rotation is

$$\frac{q_v}{s_v} + \sum_{e|v\in e} \frac{p_e x_{ev}}{s_v} + \frac{p_{e'}(\frac{\delta}{p_{e'}})}{s_v} - \frac{p_{e''}(\frac{\delta}{p_{e''}})}{s_v} = \frac{q_v}{s_v} + \sum_{e|v\in e} \frac{p_e x_{ev}}{s_v}.$$

Let us check the auxiliary constraints of LP5. Consider any edge $e = \{u, v\}$ in G where $s_u \neq s_v$. Then for e, there is an auxiliary node w_{uv} in G' with twin edges $e' = \{u, w_{uv}\}$ and $e'' = \{w_{uv}, v\}$. The orientation of both twin edges e' and e'' is determined by four variables $x_{e'u}$, $x_{e'w_{uv}}$, $x_{e''w_{uv}}$, and $x_{e''v}$ in LP5. If a twin edge is included in C, both twin edges must be in C as auxiliary nodes have degree 2. After a rotation, $x_{e'u} = x_{e'u} - \delta/p_e$, $x_{e'w_{uv}} = x_{e'w_{uv}} + \delta/p_e$, $x_{e''w_{uv}} = x_{e''w_{uv}} - \delta/p_e$, $x_{e''v} = x_{e''v} + \delta/p_e$. Therefore, $x_{e'u} = x_{e''w_{uv}}$ and $x_{e'w_{uv}} = x_{e''v}$.

Finally we prove that the tree constraint is not violated after a rotation. Our argument is based on the one given for this property by Ebenlendr *et al.* (Theorem 1 in [20]). Pick any big tree $T \in T_B$ in $G'_{\mathbf{x}}$. We show that the tree constraint is not violated for T. First, observe that if $(v, e) \in L(T)$, the value of x_{ev} changes if and only if e is in C. So if C changes the values of any variables corresponding to the endpoints of internal vertices in T, then they do not affect the tree constraint for T. Next, if a cycle C is directed towards v, the value of x_{ev} increases by δ, which is not an issue. But, when C is directed away from v, the rotation decreases the value of x_{ev} by δ, which is problematic as it decreases $\sum_{(v,e)\in L(T)} p_e x_{ev}$ by δ if C does not include another leaf of T. We will be extending T to a tree T' we define later and show that the tree constraint holds after a rotation for both T and T'.

For a vertex $t \in C$, let e_t be the edge directed away from t in C. Let W be the set of vertices $t \in T \cap C$ where e_t is not in T and t is not a leaf of T. Then, let $T' = T \cup \{e_t \mid t \in W\}$. We show that any $e_t \in T' \setminus T$ is a big edge. A rotation is applied if there are no leaf pairs in $G'_{\mathbf{x}}$, so every vertex in $G'_{\mathbf{x}}$ has degree at least 2. Since t is not a leaf in T, there is another big edge belonging to some big tree in $T_B \setminus T$ as big edges are taken in priority over small edges in $G'_{\mathbf{x}}$ and there are at least two big edges incident to t where no more than one is already part of the path that includes t. Hence, e_t is big.

By Lemma 4.2.2, the connected components of big edges in $G'_{\mathbf{x}}$ are either trees or cycles. If T is a part of a connected component of big edges that is a cycle, then $W = \emptyset$, because the algorithm will follow the cycle of the connected component of big edges through T. This implies T' is a connected subgraph of a big tree in T_B when $T' \neq T$, where each e_t is an edge incident with a leaf of T'. By the definition of e_t, edge e_t is directed towards a leaf vertex added in T' in C. Observe that all the leaves in T are also leaves of T', and the value κ_T for tree T' is the same for T. So by how T' is built, the number of leaf pairs directed towards a leaf in T' in C is at least the number of leaf pairs directed away from a leaf in T'. Hence the tree constraint for T' is preserved, and

$$\sum_{(v,e)\in L(T')} p_e x_{ev} \geq \sum_{(v,e)\in L(T')} p_e - \kappa_T.$$

Rewrite both sides of this inequality as

$$\sum_{(v,e)\in L(T)} p_e x_{ev} + \sum_{(v,e)\in L(T')\setminus L(T)} p_e x_{ev} \geq \sum_{(v,e)\in L(T)} p_e + \sum_{(v,e)\in L(T')\setminus L(T)} p_e - \kappa_T.$$

As $p_e x_{ev} \leq p_e$ for each $(v,e) \in L(T') \setminus L(T)$, then the above inequality implies that

$$\sum_{(v,e)\in L(T)} p_e x_{ev} \geq \sum_{(v,e)\in L(T)} p_e - \kappa_T.$$

Therefore, all the constraints of LP5 are preserved after a rotation. ∎

Lemma 4.2.5 *Using our rounding procedure, an orientation for G can be produced in polynomial time.*

Proof Every edge in $G'_{\mathbf{x}}$ is directed toward one of its two endpoints by either a leaf assignment, tree assignment, or a rotation. These three operations each take polynomial time and each orients at least one edge. So after a linear number of iterations, all the edges in G' are oriented. For each auxiliary node in G', by Lemma 4.2.4 both of its twin edges are directed in the same direction.

Once the rounding procedure for G' is completed, to obtain the final orientation for each $e = \{u, v\} \in E$ in G we proceed as follows:

1. if $s_u = s_v$, then orient e the same as its corresponding edge in G';
2. if $s_u \neq s_v$, then there is an auxiliary node w_{uv} with twin edges e' and e'' in G'. If e' is directed toward w_{uv} and e'' is directed to v, then orient e towards v, otherwise orient it towards u.

This takes polynomial time, therefore, we obtain an orientation for G in polynomial time. ∎

Analysis of Case 1 of Lemma 4.2.3

As stated in Chapter 4.2.4, for Case 1 of Lemma 4.2.3, $t_{\mathfrak{z}_f} = t_{\mathfrak{z}_s} = 5/6$. We show that for this case our rounding yields an orientation with makespan at most $11/6$.

Lemma 4.2.6 *Our algorithm produces an orientation with makespan no more than $11/6$ if $\ell_b/\mathfrak{z}_f \leq 1/2$ and either: (i) $\ell_b > 1$, $\Delta < \ell_s \leq 1$, and $\ell_s/\mathfrak{z}_f \leq 1/2$, or (ii) $\Delta < \ell_b \leq 1$, and $\ell_s \leq 1/2$.*

Proof Let γ' be the orientation for G produced by our algorithm. Before the rounding procedure, we have a fractional solution $\mathbf{x}$ with load on each vertex $v \in V$ at most 1. By Lemma 4.2.2 the subgraph of big edges in $G'_{\mathbf{x}}$ is a disjoint union of simple cycles and trees. Since any edges with $x_{ev} \in \{0, 1\}$ have already been oriented, we consider any load increases caused by rounding fractionally oriented edges in $G'_{\mathbf{x}}$. Once a vertex becomes isolated in $G'_{\mathbf{x}}$, its load cannot be increased any further.

We analyze the makespan of γ' based on the number of load increases for a vertex v in G'. By Lemma 4.2.4, rotations maintain all the constraints of LP5, so the load increases on the vertices are only caused by leaf assignments and tree assignments.

1. Let (v, e) be a leaf pair, where $e = \{v, z\}$. If the load of v is increased only once through a leaf assignment, then either $p_e x_{ez} \leq t_{s_v}$ or e is a small non-twin edge. Hence, the load increases on v by at most $\max\{t_{s_v}/s_v, 1/2\} \leq t_{s_v} = 5/6$, and then v becomes isolated in $G'_{\mathbf{x}}$.

2. Consider now when the load of a vertex v is increased twice: once through a tree assignment, and then by a leaf assignment. Let the tree assignment be started at vertex u' from leaf pair (u', e) where $e = \{u', z\}$. This tree assignment constructs a maximal tree $T \in T_B$ of big edges that is rooted at u' if e is big, and rooted at auxiliary node z if e is a small twin edge. The tree assignment orients e and the edges in T away from u'. Note that if any leaf l of T is an auxiliary node, the twin edge incident on l that is not in T will be directed toward some vertex v' not in T thus increasing the load of v'; we address this in Case 3 below.

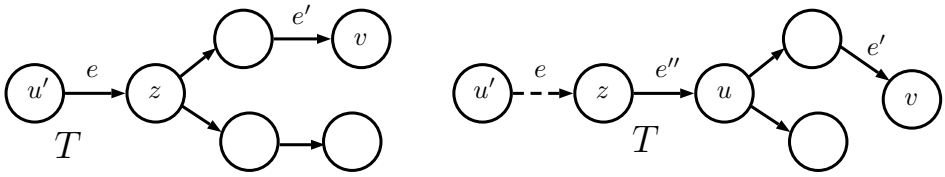

Figure 4.2: Two example situations of a tree assignment: a tree assignment that starts at a big edge (left) and a tree assignment that starts at a small twin edge (right). The path P consists of the edges from u' to v.

The path P in T from the root of T to v contains at least one big edge. Let e' be the big edge in P oriented towards v by the tree assignment. If the path P starts at auxiliary node z, let $e'' = \{z, u\}$ be the other twin edge incident on z, where $u' \neq u$ (see Figure 4.2). The auxiliary constraints guarantee that $p_{e''}x_{e''z} = p_e x_{eu'}$, $p_{e''}x_{e''u} = p_e x_{ez}$. Hence, regardless of whether the tree is rooted at z or u', by applying the tree constraint to path P, we get

$$p_e x_{eu'} + p_{e'} x_{e'v} \geq p_e + p_{e'} - \kappa_T$$

and so

$$p_{e'} - p_{e'} x_{e'v} \leq -p_e(1 - x_{eu'}) + \kappa_T = \kappa_T - p_e x_{ez}.$$

Since a tree assignment at u' is performed when $p_e x_{ez} > t_{s_{u'}}$, then

$$p_{e'} - p_{e'} x_{e'v} < \kappa_T - t_{s_{u'}}. \tag{4.8}$$

An important note is that we derived this inequality for any tree assignment in a manner so that it applies for any case of Lemma 4.2.3; we will use this inequality several times throughout our analysis.

In Case 1(a) of Lemma 4.2.3 the big edges have length $\ell_s \leq 1$ and in Case 1(b) they have length $\ell_b \leq 1$, hence $\kappa_T \leq 1$ for all $T \in T_B$. Since $t_{s_{u'}} = 5/6$, the load on v can increase by at most

$$\frac{p_{e'} - p_{e'} x_{e'v}}{s_v} < \frac{(\kappa_T - t_{s_{u'}})}{s_v} \leq \frac{(1 - \frac{5}{6})}{s_v} = \frac{1}{6s_v} \tag{4.9}$$

from the tree assignment. Next, if a leaf assignment also occurs on v, it must be from a small edge because if it were from a big edge, such an edge would be in the maximal big tree T built in the tree assignment. After the leaf assignment, the total load increase on v is at most $(1/6)/s_v + 1/2 \leq 2/3$.

3. Finally, we consider the case when the load of a vertex v is increased more than once through tree assignments and then once through a leaf assignment. Note that in this case there is at least one small twin edge e_i incident on v and on some auxiliary node w_i. The other twin edge of e_i incident on w_i must be big and thus, the vertex v must have speed $\mathfrak{s}_f$. Also since $\ell_b/\mathfrak{s}_f \leq 1/2$, v cannot be a part of a big tree in a tree assignment, and all the twin edges directed towards v by tree assignments are small (see Figure 4.3).

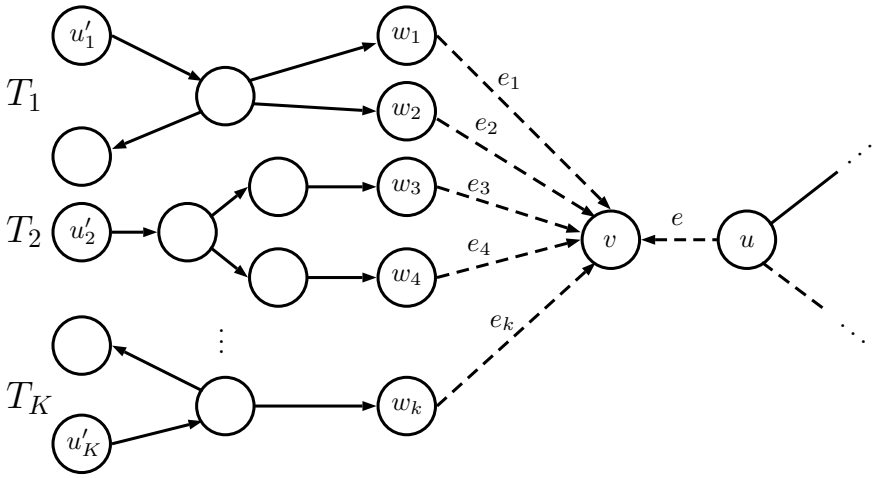

Figure 4.3: Case 3. Small edges $e_1, \ldots, e_k$ are directed towards v by tree assignments on big trees $T_1, \ldots, T_K$. At most one leaf assignment can occur on v through a small edge e. Big edges are solid lines, and small edges are dashed lines.

Let k be the number of small twin edges contributing to the load of v through tree assignments, and let K be the number of big trees involved in these tree assignments. Let $e_1, \ldots, e_k$ be the edges connecting v to those big trees through auxiliary nodes $w_1, \ldots, w_k$, and let e be the edge involved in the leaf assignment that further increases the load on v. Let the first tree assignment be performed on T_1 starting at vertex u'_1, the second on T_2 starting at vertex u'_2, and so on. Let $\mathbf{x}^{(j)}$ be the fractional solution just before the jth tree assignment is performed. Observe that the other twin edge of any w_i that is not e_i is in a path P from some u'_j to w_i. By (1) and the auxiliary constraints of each w_i, just before the first tree assignment is performed the following inequality holds

$$p_{e_i} - p_{e_i} x^{(1)}_{e_i v} < \kappa_T - t_{s_{u'_1}} \Leftrightarrow p_{e_i} x^{(1)}_{e_i v} > p_{e_i} - (\kappa_T - t_{s_{u'_1}}) \tag{4a}$$

for each e_i incident on $w_i \in T_1$. By constraint (Load v),

$$\sum_{i=1}^{k} \frac{p_{e_i} x^{(1)}_{e_i v}}{\mathfrak{s}_f} \leq 1.$$

The edges of T_1 are removed from $G'_{\mathbf{x}}$, so the value of $x^{(1)}_{e_i v}$ for each e_i incident on $w_i \in T_1$ will not change any more. Similarly, just before the second tree assignment is performed,

$$p_{e_i} x^{(2)}_{e_i v} > p_{e_i} - (\kappa_T - t_{s_{u'_2}}) \tag{4b}$$

for each e_i incident on $w_i \in T_2$. Since a rotation does not modify the load on a vertex then by constraint (Load v),

$$\sum_{e_i|w_i\in T_1} \frac{p_{e_i} x^{(1)}_{e_i v}}{\vartheta_f} + \sum_{e_i|w_i\in T_2,\dots,T_K} \frac{p_{e_i} x^{(2)}_{e_i v}}{\vartheta_f} \leq 1.$$

By the same argument, just before the last tree assignment is performed,

$$p_{e_i} x^{(K)}_{e_i v} > p_{e_i} - (\kappa_T - t_{s_{u'_K}}) \tag{4K}$$

for each e_i incident on $w_i \in T_K$, and

$$\sum_{j=1}^{K} \sum_{e_i|w_i\in T_j} \frac{p_{e_i} x^{(j)}_{e_i v}}{\vartheta_f} \leq 1.$$

Hence, by inequalities (4a), (4b), …, (4K):

$$1 \geq \sum_{j=1}^{K} \sum_{e_i|w_i\in T_j} \frac{p_{e_i} x^{(j)}_{e_i v}}{\vartheta_f} > \sum_{j=1}^{K} \sum_{e_i|w_i\in T_j} \frac{p_{e_i} - (\kappa_T - t_{s_{u'_j}})}{\vartheta_f}. \tag{4.10}$$

In case 1 of Lemma 4.2.3, every big edge has length greater than Δ. Two twin edges incident on the same auxiliary node have the same length, so each small twin edge e_i directed towards v has length $p_{e_i} > \Delta$. Since every big tree $T \in T_B$ has $\kappa_T \leq 1$, and $t_{\vartheta_s} = t_{\vartheta_f} = 5/6$,

$$1 > \sum_{j=1}^{K} \sum_{e_i|w_i\in T_j} \frac{p_{e_i} - (\kappa_T - t_{s_{u'_j}})}{\vartheta_f} \geq \sum_{j=1}^{K} \sum_{e_i|w_i\in T_j} \frac{p_{e_i} - (1 - \frac{5}{6})}{\vartheta_f}$$

$$> \sum_{i=1}^{k} \frac{(\Delta - \frac{1}{6})}{\vartheta_f} = \frac{(\Delta - \frac{1}{6})k}{\vartheta_f}.$$

Therefore

$$\vartheta_f > \Big(\Delta - \frac{1}{6}\Big)k. \tag{4.11}$$

By (4.9) and the auxiliary constraints, a tree assignment directing one small twin edge towards v increases the load of v by at most $(1/6)/\vartheta_f$. Recall that in addition a leaf assignment can orient a small edge towards v. Since there are k small twin edges, by (4.11) the total load increase on v caused by the tree assignments and a leaf assignment is at most

$$\frac{1}{6\vartheta_f} \cdot k + \frac{1}{2} < \frac{k}{6(\Delta - \frac{1}{6})k} + \frac{1}{2} = \frac{1}{6\Delta - 1} + \frac{1}{2} < \frac{1}{6(\frac{5}{6}) - 1} + \frac{1}{2} = \frac{3}{4}. \qquad \blacksquare$$

Analysis of Case 2 of Lemma 4.2.3

Recall that for Case 2 of Lemma 4.2.3, we set $t_{\vartheta_f} = t_{\vartheta_s} = (3\ell_b)/4$.

Lemma 4.2.7 *Our algorithm produces an orientation with makespan at most* $1+\Delta$ *when* $1/2 < \ell_b/\vartheta_f \leq 1$ *and either: (i)* $\ell_b > \Delta$*, and* $\ell_s \leq 1/2$*, or (ii)* $\ell_b > 1$*,* $1/2 < \ell_s \leq \Delta$*, and* $\ell_s/\vartheta_f \leq 1/2$*.*

Proof First we consider Case 2(a) i.e. $1/2 < \ell_b/\vartheta_f \leq 1$, $\ell_b > \Delta$, and $\ell_s \leq 1/2$. Note that every big edge has length ℓ_b, and so $\ell_b = \kappa_T$ for all big trees $T \in T_B$. Since $\ell_b/\vartheta_f > 1/2$, a tree assignment always constructs a maximal tree where none of the leaves are auxiliary nodes. This implies that the load of a vertex cannot be increased more than once by a tree assignment, which simplifies our analysis.

1. If a leaf assignment on leaf pair (v, e) where $e = \{v, z\}$ increases the load of vertex v, then if e is big, $p_e x_{ez} \leq (3\ell_b)/4$, and the load increase on v is at most $(3\ell_b)/(4s_v) \leq 3/4$; otherwise e is small, and the load increase on v is at most $1/2$.

2. Next, we consider when the load of a vertex $v \in V$ is increased twice: once through a tree assignment caused by leaf pair (u', e) where $e = \{u', z\}$ has length ℓ_b, and once by a leaf assignment. Since $p_e x_{ez} > (3\ell_b)/4$, by Inequality (4.8), the load increase on v from a tree assignment is at most

$$\frac{p_{e'} - p_{e'} x_{e'v}}{s_v} < \frac{\kappa_T - t_{s_{u'}}}{s_v} = \frac{\ell_b - \frac{3\ell_b}{4}}{s_v} = \frac{\ell_b}{4s_v} \leq \frac{1}{4}.$$

   Additionally a small edge can be oriented towards v by a leaf assignment. Thus, the total load increase on v is at most $1/4 + 1/2 = 3/4$.

Now we analyze Case 2(b) i.e. $1/2 < \ell_b/\vartheta_f \leq 1$, $\ell_b > 1$, $1/2 < \ell_s \leq \Delta$, $\ell_s/\vartheta_f \leq 1/2$. Since $\ell_s > 1/2$, from the remark in Lemma 4.2.3 we get $\Delta < (3\ell_b)/4$. Notice that each connected component formed by big edges in G' contains either edges of length ℓ_s incident on vertices with speed 1 or edges of length ℓ_b incident on vertices with speed ϑ_f. Similar to Case 2(a), a leaf pair (v, e) with $p_e = \ell_s$ cannot cause a tree assignment as $\ell_s \leq \Delta < (3\ell_b)/4$. Edges with length ℓ_b are only incident on vertices with speed ϑ_f; hence, a tree assignment on a big tree with edges of length ℓ_b does not orient any small twin edges away from auxiliary nodes, and so the load of a vertex cannot be increased several times by tree assignments. There are two situations that need to be considered.

1. The load of a vertex v is increased by a leaf assignment. A leaf assignment that orients a big edge increases the load of v by at most $3/4$ if $s_v = \vartheta_f$, and by no more than Δ when $s_v = 1$. A leaf assignment that orients a small edge increases the load of v by at most $1/2$.

2. The other possibility is that a tree assignment plus a leaf assignment increase the load of a vertex v with speed ϑ_f. Let $e = \{u', z\}$ be the edge directed away from the vertex u' in the tree assignment caused by leaf pair (u', e) that increases the load of v. Also let e' be the

edge directed towards v by the tree assignment. Since $p_e x_{ez} > (3\ell_b)/4$, Inequality (4.8) implies the load increase on v from the tree assignment is at most

$$\frac{p_{e'} - p_{e'} x_{e'v}}{\vartheta_f} \leq \frac{\kappa_T - p_e x_{ez}}{\vartheta_f} < \frac{\ell_b - \frac{3\ell_b}{4}}{\vartheta_f} = \frac{\ell_b}{4\vartheta_f} \leq \frac{1}{4}.$$

A small edge can then be oriented towards v, so the total load increase on v is at most $3/4$.

∎

Analysis of Case 3 of Lemma 4.2.3

Recall that in Case 3 of Lemma 4.2.3, $1/2 < \ell_b/\vartheta_f \leq 1$, $\ell_b > 1$, $1/2 < \ell_s \leq 1$, and $\ell_s/\vartheta_f \leq 1/2$ and we set $t_{\vartheta_s} = \Delta$ and $t_{\vartheta_f} = (3\ell_b)/4$.

Lemma 4.2.8 *Our algorithm produces an orientation with makespan at most* $1+\Delta$ *when* $1/2 < \ell_b/\vartheta_f \leq 1$, $\ell_b > 1$, $\Delta < \ell_s \leq 1$, *and* $\ell_s/\vartheta_f \leq 1/2$.

Proof Similar to Case 2(b), each connected component formed by big edges in G' contains either only edges of length ℓ_s incident on vertices with speed 1 or only edges of length ℓ_b incident on vertices with speed ϑ_f. Now, we bound the total load increase of each vertex.

1. If a leaf assignment orients a big edge towards a leaf v, then the load increase is at most $t_{\vartheta_s} = \Delta$ if $s_v = 1$ and $(3\ell_b)/(4\vartheta_f) \leq 3/4$ if $s_v = \vartheta_f$. If a leaf assignment directs a small edge towards a leaf v, this causes a load increase of at most $1/2$ on v.

2. The load of a vertex v can be increased twice: once through a tree assignment and then through a leaf assignment. The setup in this case is the same as Case 2 in the proof of Lemma 4.2.6. Let T be the maximal tree built in the tree assignment that caused the first load increase on v, and let e' be the edge directed towards v by the tree assignment. Observe that T either consists only of vertices with speed ϑ_f or T contains only vertices with speed 1. If all the vertices in T have speed $s_{u'}$ and $e = \{u', z\}$, applying Inequality (4.8) and that $p_e x_{ez} > t_{s_{u'}}$ we get

$$p_{e'} - p_{e'v} < \kappa_T - t_{s_{u'}}. \tag{4.12}$$

If v has speed 1, then $\ell_s = \kappa_T$, and by (4.12) the load increase on v from the tree assignment is at most $\ell_s - t_{\vartheta_s} \leq 1 - t_{\vartheta_s}$. Otherwise, v has speed ϑ_f, and $\ell_b = \kappa_T$, so by (4.12) the load increase on v is at most $(\ell_b - (3\ell_b)/4)/\vartheta_f = \ell_b/(4\vartheta_f) \leq 1/4$.

Additionally a leaf assignment can orient an edge towards v. Since $v \in T$ and T is a maximal tree formed by big edges, the edge directed towards v by the leaf assignment must be small. Thus, the total load increase on v is at most $1 - t_{\vartheta_s} + 1/2 \approx 0.617217$ if $s_v = 1$, and at most $1/4 + 1/2 = 3/4$ if $s_v = \vartheta_f$.

3. It is also possible that several tree assignments direct small twin edges towards a vertex v with speed ϑ_f. Observe that any edge incident on v that is small has length ℓ_s, hence such tree assignments must be started at leaf vertices with speed ϑ_s. Unlike in the proof of Lemma 4.2.6, either a tree assignment can still occur with big edges of length ℓ_b that

further increases the load of v plus a leaf assignment that directs a small edge toward v, or a leaf assignment can direct a big edge of length ℓ_b towards v.

Let $e_1, e_2, \ldots, e_k$ be small twin edges that are directed towards v from tree assignments; these twin edges are incident on auxiliary nodes $w_1, \ldots, w_k$ connected to $K \geq 1$ big trees $T_1, \ldots, T_K$ (see Figure 4.3). Let the algorithm perform the first tree assignment on T_1, the second on T_2, and so on. Also, let $\mathbf{x}^{(j)}$ be the fractional solution right before the jth tree assignment. Using the same argument in Lemma 4.2.6 for Case 3 and that the big trees $T_1, \ldots, T_K$ all have edges of length $\ell_s = \kappa_T$, we get

$$1 \geq \sum_{j=1}^{K} \sum_{e_i | w_i \in T_j} \frac{p_{e_i} x_{e_i v}^{(j)}}{\mathcal{s}_f} > \sum_{j=1}^{K} \sum_{e_i | w_i \in T_j} \frac{p_{e_i} - (\kappa_T - t_{\mathcal{s}_s})}{\mathcal{s}_f}$$

$$= \sum_{i=1}^{k} \frac{\ell_s - (\ell_s - t_{\mathcal{s}_s})}{\mathcal{s}_f} = \frac{t_{\mathcal{s}_s} k}{\mathcal{s}_f},$$

so $\mathcal{s}_f > t_{\mathcal{s}_s} k$. By (4.12) and the auxiliary constraints, one tree assignment directing a small twin edge towards v increases the load of v by at most $(1 - t_{\mathcal{s}_s})/\mathcal{s}_f$. Thus, when k small twin edges are directed towards v, the total load increase on v is at most

$$\frac{(1 - t_{\mathcal{s}_s})k}{\mathcal{s}_f} < \frac{(1 - t_{\mathcal{s}_s})k}{t_{\mathcal{s}_s} k} = \frac{1}{t_{\mathcal{s}_s}} - 1.$$

Now, we must consider two situations.

(a) A leaf assignment orients an edge toward v. This edge may be big or small.
   (i) If the edge is small, the total load increase on v is at most $(1/t_{\mathcal{s}_s} - 1) + 1/2 \approx 0.63278$.
   (ii) If the edge is big it has length ℓ_b. Then, the total load increase on v is no more than $1/t_{\mathcal{s}_s} - 1 + 3/4 = \Delta$.

(b) A tree assignment orients a big edge of length ℓ_b towards v, then a leaf assignment directs a small edge towards v. Since the tree assignment involves vertices of speed $\mathcal{s}_f$, the total load increase is at most $(1/t_{\mathcal{s}_s} - 1) + (1 - 3/4) + 1/2 = 1/t_{\mathcal{s}_s} - 3/4 + 1/2 = \Delta$. ∎

Theorem 4.2.9 *There is a* $(\sqrt{65}+7)/8$*-approximation algorithm for the graph balancing problem with two speeds and two lengths.*

4.3 Integrality Gaps for Linear Programs LP5 and LP6

In this section, we discuss integrality gaps for LP5 and LP6. While there are integrality gap results for linear program formulations of the unrelated graph balancing problem and the graph balancing problem with speeds ([20, 97]), these integrality gap results do not cover the case when there are only two speeds. For arbitrary numbers of machine speeds and job lengths Ebenlendr *et al.* [20] showed that LP2 has integrality gap 2 for the graph balancing problem

with speeds, and Vershae and Wiese [97] proved that LP2 has integrality gap 2 for the unrelated graph balancing problem when there are processing times $p_{i,j} \in \{\epsilon, 1, \infty\}$ for some $\epsilon > 0$. In this section we show that linear programs LP5 and LP6 from Chapter 4.2 have integrality gaps at least $5/3$ and $7/4$ when there are two job lengths and three job lengths, respectively; we use an instance similar to the one derived for LP3 by Ebenlendr *et al.* [20].

Theorem 4.3.1 *The integrality gap of LP5 for the graph balancing problem with two speeds and two job lengths is at least* $5/3$.

Proof Let $0 < \epsilon \leq 1/3$. Build the weighted multigraph G' shown in Figure 4.4 with two speeds $s_2 = 2 - 3\epsilon$ and $s_1 = 1$. Let $L > 1$, and the number of edges from each vertex a_i to vertex b_i is $2L + 1$. Note that G' is comprised of three vertex-disjoint paths of equal but odd length connected at two vertices, u and u', such that the middle edge of each path is small.

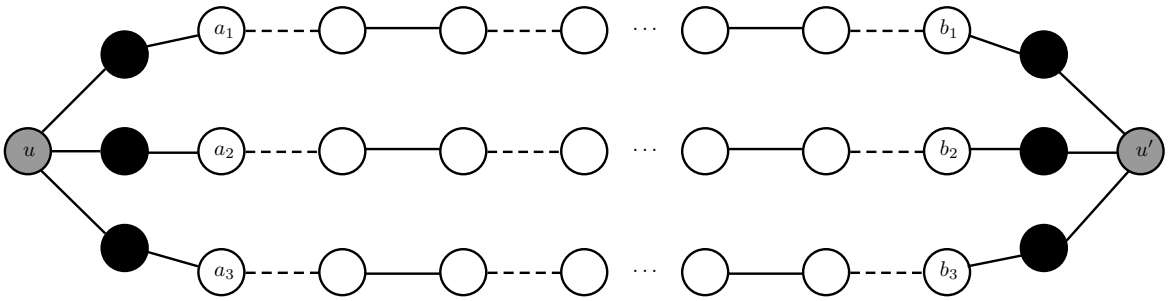

Figure 4.4: Multigraph G'. Solid lines represent edges of length $1 - \epsilon$, the dashed lines are edges of length $1/3$, and the black vertices are auxiliary nodes. The lightly-shaded vertices u and u' have speed $2 - 3\epsilon$, and all others have speed 1. The auxiliary nodes have dedicated load 0, and every other vertex has dedicated load $1/3$.

First we show that there is a valid solution for LP5 assuming the three paths connected to u and u' in G' are of sufficient length. Let $c = (2 - 3\epsilon)/(6 - 6\epsilon)$. For each big edge e' incident on u or u', set $x_{e'u} = c$ and $x_{e'u'} = c$, and then for each auxiliary node w adjacent to u and u', set $x_{e'w} = (1 - c)$. To preserve the auxiliary constraints, for each twin edge $e'' = \{w, a_i\}$ or $e'' = \{w, b_i\}$, $i = 1, 2, 3$, set $x_{e''w} = c$ and $x_{e''z} = (1 - c)$. Next, consider the remaining edges and vertices in each of the three vertex-disjoint paths. Let $e_m = \{l_0, r_0\}$ be the middle edge in one of these paths P_i; as mentioned above, e_m has length $1/3$. Let $d \geq 0$, and let l_d and r_d be vertices that are d vertices away from l_0 and r_0, respectively and let $e_{l_d} = \{l_{d+1}, l_d\}$ and $e_{r_d} = \{r_d, r_{d+1}\}$. Since in P_i the number of edges from a_i to b_i is $2L + 1$ then $l_L = a_i$ and $r_L = b_i$. Assign $x_{e_m l_0} = 1/2$ and $x_{e_m r_0} = 1/2$, and then set $x_{e_{l_0} l_0} = (1/2)/(1 - \epsilon)$ and $x_{e_{r_0} r_0} = (1/2)/(1 - \epsilon)$. Now, for each $d = 0, \ldots, L - 1$, assign

$$x_{e_{l_d} l_{d+1}} = (1 - x_{e_{l_d} l_d}),\ x_{e_{l_{d+1}} l_{d+1}} = \min\left\{1, \frac{1 - (p_{e_{l_d}} x_{e_{l_d} l_{d+1}} + \frac{1}{3})}{p_{e_{l_{d+1}}}}\right\},$$

and set

$$x_{e_{r_d} r_{d+1}} = (1 - x_{e_{r_d} r_d}),\ x_{e_{r_{d+1}} r_{d+1}} = \min\left\{1, \frac{1 - (p_{e_{r_d}} x_{e_{r_d} r_{d+1}} + \frac{1}{3})}{p_{e_{r_{d+1}}}}\right\}.$$

Observe that the big edges e_{l_0} and e_{r_0} are each oriented by more than half toward l_0 and r_0 respectively as $\epsilon > 0$. Then, small edges e_{l_1} and e_{r_1} are fractionally oriented by more than half toward l_1 and r_1 respectively, as

$$x_{e_{l_0}l_1} = \left(1-\frac{1}{2(1-\epsilon)}\right) \text{ and } x_{e_{l_1}l_1} = \left(\frac{1-\left((1-\epsilon)(1-\frac{1}{2(1-\epsilon)})+\frac{1}{3}\right)}{\frac{1}{3}}\right) = 3\left(1-(1-\epsilon)+\frac{1}{2}-\frac{1}{3}\right) = \frac{1}{2}+3\epsilon;$$

the values are similar for $x_{e_{r_0}r_1}$ and $x_{e_{r_1}r_1}$. Likewise, since an additional $3\epsilon/(1/3) = \epsilon$ amount of load was contributed to l_1 and r_1 by small edges e_{l_1} and e_{r_1}, again, the big edges e_{l_2} and e_{r_2} are oriented by more than half toward l_2 and r_2, which leaves an additional 2ϵ of load free on l_3 and r_3 for small edges; this allows for small edges e_{l_3} and e_{r_3} to be fractionally oriented by $1/2 + 6\epsilon$ toward l_3 and r_3, respectively. Furthermore, each small edge e_{l_d} has $x_{e_{l_d}l_d} = 1/2 + \lceil d/2 \rceil(3\epsilon)$, which holds similarly for e_{r_d}. Thus, if L is sufficiently large so that $\lceil (L-1)/2 \rceil(3\epsilon) \geq 1/2$, the small edges $e_{l_{L-1}}$ and $e_{r_{L-1}}$ incident on a_i and b_i are oriented away from vertices l_L and r_L and toward l_{L-1} and r_{L-1}, respectively.

Clearly all the edge constraints are satisfied. Let us check the load constraints associated with each vertex. First, the load of either u or u' is

$$\frac{\frac{1}{3}}{2-3\epsilon} + \frac{3c(1-\epsilon)}{2-3\epsilon} = \frac{1}{6-9\epsilon} + \frac{3(2-3\epsilon)(1-\epsilon)}{(6-6\epsilon)(2-3\epsilon)} = \frac{1}{6-9\epsilon} + \frac{3(1-\epsilon)}{6(1-\epsilon)} \leq \frac{1}{2} + \frac{1}{2} = 1.$$

Next, the vertices a_i, b_i, $i = 1, 2, 3$ each have load

$$\frac{1}{3} + (1-\epsilon)(1-c) = \frac{1}{3} + (1-\epsilon)\left(1 - \frac{2-3\epsilon}{6-6\epsilon}\right) = \frac{1}{3} + \frac{(1-\epsilon)(4-3\epsilon)}{6(1-\epsilon)} = \frac{1}{3} + \frac{4-3\epsilon}{6} < 1.$$

It is not hard to see from the way the values of the variables are assigned for the edges incident on the remaining vertices of the three paths that their loads are each at most 1.

Consider now any big tree $T \in T_B$. Notice that every big edge in T has length $1 - \epsilon = \kappa_T$. First, if u or u' is a leaf in T, then there is only one other leaf in T and that leaf is incident on a twin edge and so

$$\sum_{(v,e)\in L(T)} p_e x_{ev} = c(1-\epsilon) + (1-c)(1-\epsilon) = 1 - \epsilon = \sum_{(v,e)\in L(T)} p_e - \kappa_T.$$

If a big tree consists of a single big edge along one of the three paths, because the edge constraints hold, the tree constraints for any such big tree are satisfied. Otherwise, the leaves are either auxiliary nodes or vertices that are neither u nor u' but are adjacent to auxiliary nodes. Let the number of leaves in a big tree T be k. As the degree of any vertex in T is at most 3, $k \leq 3$. As $(k(2-3\epsilon))/(6-6\epsilon) < 1$ for $0 \leq k \leq 3$, then

$$\sum_{(v,e)\in L(T)} p_e x_{ev} = (1-\epsilon)\Big(k(1-c)\Big) = (1-\epsilon)\left(k - \frac{k(2-3\epsilon)}{6-6\epsilon}\right)$$

$$> (1-\epsilon)(k-1) = \sum_{(v,e)\in L(T)} p_e - \kappa_T.$$

Thus, all the constraints of LP5 are satisfied, so the makespan of this solution is at most 1.

Now we determine the optimal integral solution to LP5. Due to the tree constraints and the auxiliary constraints, at most one big twin edge can be oriented toward u and u', so the loads of u and u' are at most $(1/3)/(2-3\epsilon) + (1-\epsilon)/(2-3\epsilon) \leq 1$. Along the three paths from a_i to b_i, $i = 1, 2, 3$, observe that at least one vertex in one of the three paths must have at least two edges oriented towards it. The speed of any of these vertices is 1, so the makespan is $(1-\epsilon)+1/3+1/3 = 5/3-\epsilon$. Therefore, if we take $\epsilon \to 0$, LP5 has integrality gap at least $5/3$ for the graph balancing problem with two speeds and two job lengths. ∎

In the above proof it is not hard to see that the star constraint of LP6 is also satisfied, so our example above works also for LP6.

Corollary 4.3.2 *For the graph balancing problem with two speeds and two job lengths, the integrality gap of LP6 is at least* $5/3$.

To close this section, we consider LP5 and LP6 for the graph balancing problem with two speeds and three job lengths. Build the same weighted multigraph G' we did in Theorem 4.3.1 with the following modifications. First, the dedicated loads of all the vertices except the auxiliary nodes are $1/4$, and all auxiliary nodes have no dedicated load. Next, the edges with length $1-\epsilon$ now have length 1, and the edges with length $1/3$ have length $1/2-\epsilon$. Finally, set the speeds $s_2 = 2-\epsilon$ and $s_1 = 1$. An argument similar to the one given above can be used to show the integrality gap is at least $7/4$. Note that this instance extends the one in [20] for the graph balancing problem with one speed to the case of two speeds and three job lengths.

Theorem 4.3.3 *For the graph balancing problem with two speeds and three job lengths, the integrality gap of LP5 is at least* $7/4$.

Corollary 4.3.4 *For the graph balancing problem with two speeds and three job lengths, the integrality gap of LP6 is at least* $7/4$.

Chapter 5

Simple Job-Intersection Structure and Bounded Job Assignments

In this chapter we present results from our study of $R||C_{max}$ with simple job-intersection structure and $R||C_{max}$ with bounded job assignments. For any instance of $R||C_{max}$, the job-intersection graph $G_J = (J, E_J)$ is such that E_J contains edge $\{J_j, J_{j'}\}$ if there is a machine $M_i \in M$ such that $p_{i,j} \neq \infty$ and $p_{i,j'} \neq \infty$. Let $\mathcal{J}_i$ be the set of jobs that can be scheduled on machine M_i. Recall that in $R||C_{max}$ with simple job-intersection structure we restrict the problem to instances of $R||C_{max}$ where G_J belongs to a specific graph class, and in $R||C_{max}$ with bounded job assignments $|\mathcal{J}_i| \leq \varkappa$ for some fixed $\varkappa > 0$ for every $M_i \in M$. We focus on instances of $R||C_{max}$ of both these problems where there are only a few eligible jobs per machine.

In Chapter 5.1 we show that there are no ρ-approximation algorithms with $\rho < 3/2$ for both $R||C_{max}$ restricted to diamondless job-intersection graphs and $R||C_{max}$ with bounded job assignments when $\varkappa = 3$, unless $\mathsf{P} = \mathsf{NP}$; while these results match the best-known $3/2$ lower bound for $R||C_{max}$, these results are stronger hardness results than previously known. In Chapter 5.2 we present a flow-based approximation algorithm that is a $5/3$-approximation algorithm for $R||C_{max}$ with $\varkappa = 4$, a $3/2$-approximation algorithm for both $R||C_{max}$ with $\varkappa = 3$ and $R||C_{max}$ restricted to diamondless job-intersection graphs, and a polynomial-time algorithm for both $R||C_{max}$ with $\varkappa = 2$ and $R||C_{max}$ restricted to triangle-free job-intersection graphs. In Chapter 5.3 we give a $(2 - 1/(\varkappa - 1))$-approximation algorithm for the restricted assignment problem with two job lengths when $\varkappa \geq 3$. Finally, we wrap up this chapter with Chapter 5.4 by summarizing $3/2$-hardness results for $R||C_{max}$ restricted to various special classes of job-intersection graphs: complete graphs, threshold graphs, interval graphs, cographs, split graphs, and house-free graphs.

5.1 Hardness of Approximation

In this section we prove under the assumption that $\mathsf{P} \neq \mathsf{NP}$ that $R||C_{max}$ is $3/2$-inapproximable when restricted to instances with diamondless job-intersection graphs. To do this, we use the reduction from Chapter 2.2.3 (Page 25) used to show that there is no ρ-approximation algorithm with $\rho < 3/2$ for the graph balancing problem with job lengths either 1 or 2, unless $\mathsf{P} = \mathsf{NP}$. We can make a couple of simple but stronger assumptions than previously assumed for At-

Most-3-SAT(2L): Without loss of generality we assume that no clause contains a tautology, and that no clause contains duplicate literals. In addition, we will treat the dedicated load of any vertex v of value q_v as q_v self-loops of length 1.

Here we describe the reduction with our changes. Given an instance of At-Most-3-SAT(2L) where no clause in formula ϕ contains duplicate literals nor any tautologies, we build a weighted multigraph G as follows. Create one vertex for each clause y_i, and one vertex for each literal (x_i and $\neg x_i$) of every variable x_i; again, these are called clause vertices and literal vertices, respectively. For each variable x_i, add a tautologous edge $\{x_i, \neg x_i\}$ of length 2, and add $3 - |y_i|$ self-loops of length 1 to clause vertex y_i, where $|y_i|$ is the number of literals in clause y_i. Finally, for each clause y_i and literal l, add clause edge $\{l, y_i\}$ if literal l is in clause y_i.

Recall that the diamond graph is the complete graph K_4 less one edge, and that a diamondless graph contains no subgraph that is the diamond graph. We prove that any job-intersection graph G_J of an instance produced by this reduction is diamondless.

Lemma 5.1.1 *The job-intersection graph G_J of the weighted multigraph G produced by the above reduction contains no diamonds.*

Proof Every vertex in G has at most three incident edges, so G_J can only be comprised of isolated job vertices, paths, or triangles. Observe that then, there must be two triangles in G_J that share two job vertices to form a diamond. We show that no two such triangles exist in G_J.

First, consider the edges incident on literal vertices in G. Recall that each variable appears in at most three clauses in formula ϕ and each literal for that variable appears at most twice. So the job vertex corresponding to the tautologous edge $\{x_i, \neg x_i\}$ has degree at most three in G_J. Furthermore, this job vertex is only adjacent to job vertices that are clause edges in G, and at most two clause edges may have the same literal vertex as an endpoint in G. Thus, any job vertex $\{x_i, \neg x_i\}$ along with its adjacent job vertices for clause edges form in G_J either a path with one edge, a path with two edges, a triangle, or a path with an edge plus a triangle, but not two triangles (i.e. a bowtie or a diamond).

Next consider the edges incident on clause vertices in G. As no two clause vertices have edges in common in G and every clause vertex has three edges incident on it, the edges incident on the clause vertex form a triangle in G_J. Thus, if there is a diamond in G_J it must be comprised of job vertices of two clause edges that are adjacent to the job vertices of a tautologous edge and another clause edge. There is no diamond when these two clause edges are incident on two literal vertices of different variables, hence, there are two possibilities:

1. A diamond formed by two clause edges that are incident on the same literal vertex in G. This cannot happen as no clause in formula ϕ has duplicate literals.

2. A diamond formed by two clause edges that are incident on two literal vertices for the same variable in G. No clause contains tautologies, thus this situation cannot occur.

Therefore, G_J contains no diamonds. ∎

Bringing together Theorem 2.2.2 and Lemma 5.1.1, it follows that if there were a ρ-approximation algorithm with $\rho < 3/2$ for the graph balancing problem with two job lengths when G_J is diamondless, one could apply the above reduction and then use such a ρ-approximation algorithm to correctly decide whether ϕ is satisfiable or not in polynomial time.

Theorem 5.1.2 *There is no ρ-approximation algorithm with $\rho < 3/2$ for the graph balancing problem with two job lengths when the job-intersection graph G_J contains no diamonds, unless* P = NP.

The graph balancing problem is a special case of $R||C_{max}$, therefore we obtain the following corollary.

Corollary 5.1.3 *There is no ρ-approximation algorithm with $\rho < 3/2$ for $R||C_{max}$ restricted to diamondless job-intersection graphs, unless* P = NP.

If $|\mathcal{J}_i| > 3$ for some machine M_i, then there are at least four jobs J_1, J_2, J_3, J_4 such that $p_{i,1} \neq \infty$, $p_{i,2} \neq \infty$, $p_{i,3} \neq \infty$, and $p_{i,4} \neq \infty$. This would imply G_J contains a diamond; thus, for any machine M_i, $|\mathcal{J}_i| \leq 3$ is satisfied if G_J is diamondless. Hence, the case for $R||C_{max}$ when the job-intersection graph is diamondless is a special case of $R||C_{max}$ when $|\mathcal{J}_i| \leq 3$ for each machine M_i. Thus our hardness results carry over to the special case where every machine can process at most three jobs. Do note that this result can also be trivially obtained by observing that every vertex in the graph balancing instance from the reduction above has at most three incident edges.

Corollary 5.1.4 *There is no ρ-approximation algorithm for the graph balancing problem with two job lengths when every machine can process at most three jobs where $\rho < 3/2$, unless* P = NP.

5.2 Approximation Results for Unrelated Scheduling

As we stated at the end of the previous section, $R||C_{max}$ restricted to diamondless job-intersection graphs is a special case of $R||C_{max}$ when every machine M_i satisfies $|\mathcal{J}_i| \leq 3$. Here we present a $5/3$-approximation algorithm for $R||C_{max}$ when every machine can process at most four jobs. In our analysis we show the same approximation algorithm has approximation ratio $3/2$ in the case when every machine can process at most three jobs and it solves the problem exactly if $|\mathcal{J}_i| \leq 2$. Note that for $R||C_{max}$ restricted to triangle-free job-intersection graphs, $|\mathcal{J}_i| \leq 2$ for each machine M_i as if three jobs share a common machine then the job-intersection graph would have a triangle.

Given makespan estimate τ, our algorithm is a $5/3$-relaxed decision procedure when $|\mathcal{J}_i| \leq 4$. We say a job J_j is *small* on machine M_i if its processing time is $p_{i,j} \leq \tau/2$, and is *big* on machine M_i if $\tau/2 < p_{i,j} \leq \tau$. Observe that if $OPT \leq \tau$, at most one big job can be scheduled on a machine. Note that by our definitions, if $p_{i,j} > \tau$ then job J_j is neither big nor small with respect to its processing time on machine M_i. Our algorithm is described below.

1. For each machine M_i, if any job J_j has a processing time $p_{i,j} > \tau$ on M_i, we remove job J_j from job set $\mathcal{J}_i$. As a result, every job J_j in each job set $\mathcal{J}_i$ has processing time $p_{i,j} \leq \tau$.

2. Build a single-source single-sink flow network N with source s^* and sink t^*. In this network, create a job node for each job, and add arcs from s^* to each job node with capacity 1. Now, for each machine M_i, we create a machine node and a buffer node with arcs according to the *Machine Plan* given in Figure 5.1. Let disjoint sets $\mathcal{S}_i, \mathcal{B}_i \subseteq \mathcal{J}_i$,

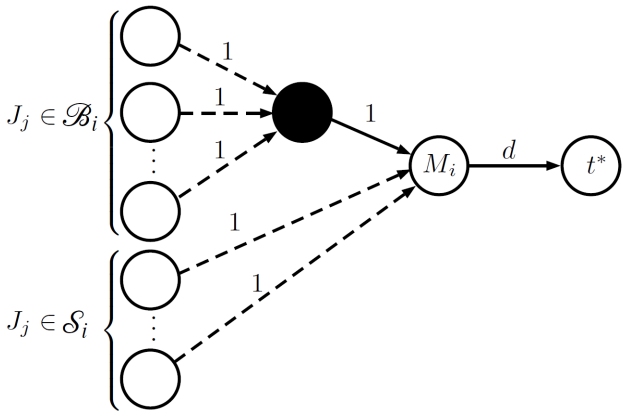

Figure 5.1: Machine Plan. Flow network N is built in part by determining the appropriate machine plan for each machine. Assume an integer value d is provided. Unlabelled white nodes are job nodes, the black node is a *buffer* node of machine M_i that only allows one unit of flow to be sent from job nodes in $\mathscr{B}_i$, and t^* is the sink of N. It is worth noting that if $\mathscr{B}_i = \emptyset$, the buffer node for the machine plan of M_i has no incoming arcs.

where unless otherwise stated $\mathscr{S}_i$ and $\mathscr{B}_i$ are the small jobs and big jobs in $\mathscr{J}_i$, respectively. Consider the following cases in the order provided:

(a) If $|\mathscr{J}_i| = 0$, no arcs are added for machine node M_i.

(b) If $\sum_{J_j \in \mathscr{J}_i} p_{i,j} \leq \tau$, then every job $J_j \in \mathscr{J}_i$ can be scheduled on machine M_i, so we add arcs according to the Machine Plan with $d = |\mathscr{J}_i|$ and set $\mathscr{S}_i = \mathscr{J}_i$ and $\mathscr{B}_i = \emptyset$.

(c) If $|\mathscr{J}_i| \leq 3$, then use the Machine Plan with $d = |\mathscr{J}_i| - 1$.

For the next set of cases $|\mathscr{J}_i| = 4$. Sort the jobs of each set $\mathscr{J}_i$ in non-increasing order by processing time; let these jobs be indexed as $j_1^{(i)}, j_2^{(i)}, j_3^{(i)}, j_4^{(i)}$.

(d) If $\sum_{k=2}^{4} p_{i,j_k^{(i)}} > \tau$, add arcs according to the Machine Plan with $d = 2$.

(e) If $\sum_{k=2}^{4} p_{i,j_k^{(i)}} \leq \tau$ and $p_{i,j_1^{(i)}} + p_{i,j_2^{(i)}} > \tau$, put jobs $j_1^{(i)}$ and $j_2^{(i)}$ into $\mathscr{B}_i$ (if either is not already there set $\mathscr{S}_i = \mathscr{J}_i \setminus \mathscr{B}_i$) and use the Machine Plan with $d = 3$.

(f) If $\sum_{k=2}^{4} p_{i,j_k^{(i)}} \leq \tau$ and $p_{i,j_1^{(i)}} + p_{i,j_2^{(i)}} \leq \tau$, use the Machine Plan with $d = 3$.

3. After N is constructed, the algorithm computes an integral maximum flow f on N. If any arc leaving the source does not carry one unit of flow, report `FAIL`; otherwise, we build a schedule as follows: for each job node J_j, if machine node M_i receives 1 unit of flow from J_j, schedule job J_j on machine M_i.

We set the value d in each case above so that if a schedule of makespan τ exists, we permit a sufficient amount of flow to reach the sink and saturate the arcs incident on the source. In addition, a schedule with makespan at most τ allows at most one job from $\mathscr{B}_i$ onto each machine M_i, so any flow carried from the source through a job node of $\mathscr{B}_i$ can be sent through the buffer node and to the sink. Thus, by the way we designed the flow network, it is not hard to

see that if $OPT \leq \tau$, all the arcs leaving the source are saturated, and as a result, a schedule is produced.

Now we analyze the load of each machine. First, it is trivial to observe that the load of any machine M_i is at most τ if either $\sum_{J_j \in \mathcal{J}_i} p_{i,j} \leq \tau$ (Case (b)) or all the jobs in $\mathcal{J}_i$ are big (Case (c) if $|\mathcal{J}_i| \leq 3$, Case (d) if $|\mathcal{J}_i| = 4$). Thus, we consider each machine M_i when there is at least one small job and $\sum_{J_j \in \mathcal{J}} p_{i,j} > \tau$ by the number of jobs in $\mathcal{J}_i$:

- $|\mathcal{J}_i| \leq 2$. If $|\mathcal{J}_i| \leq 1$, then either Case (a) or Case (b) occurs, which we already considered above. If $|\mathcal{J}_i| = 2$ and all the jobs are small, then $\sum_{J_j \in \mathcal{J}_i} p_{i,j} \leq \tau/2 + \tau/2 = \tau$. If $|\mathcal{J}_i| = 2$ and $\sum_{J_j \in \mathcal{J}_i} p_{i,j} > \tau$, then the algorithm applies Case (c), which permits only $|\mathcal{J}_i| - 1 = 1$ job to be scheduled on machine M_i, and so the load of M_i is at most τ. Therefore, the load of any machine with $|\mathcal{J}_i| \leq 2$ is at most τ.

- $|\mathcal{J}_i| = 3$. If $\sum_{J_j \in \mathcal{J}_i} p_{i,j} > \tau$, Case (c) is applied and the Machine Plan allows at most $|\mathcal{J}_i| - 1 = 2$ jobs to be scheduled on machine M_i. If all three jobs in $\mathcal{J}_i$ are small, then at most two jobs are scheduled on machine M_i and the load is at most τ. Otherwise, at least one job is big and at most two jobs are small in $\mathcal{J}_i$; at most one big job will be scheduled with a small job and so the load is at most $\tau + \tau/2 = (3/2)\tau$. Therefore, the load of any machine with $|\mathcal{J}_i| = 3$ is at most $(3/2)\tau$.

- $|\mathcal{J}_i| = 4$. First, we make a few key observations that will simplify our analysis. If Case (d) is applied then $d = 2$ in the Machine Plan, so at most one big job is scheduled with one small job and the load is at most $\tau + \tau/2 = (3/2)\tau$. Thus we only need to consider the algorithm in situations when it applies Case (e) or Case (f). In either of these two cases, $d = 3$, so at most one big job is scheduled with two small jobs, as when three small jobs are scheduled on machine M_i the load is at most $(3/2)\tau$. Recall that the jobs in $\mathcal{J}_i$ are sorted in non-increasing order of processing time: $j_1^{(i)}$, $j_2^{(i)}$, $j_3^{(i)}$, $j_4^{(i)}$. If there are at least three big jobs and at most one small job, then $\sum_{k=2}^{4} p_{i,j_k^{(i)}} > \tau$ and this falls under Case (d); thus we only need to consider below when there are at most two big jobs and at least two small jobs in $\mathcal{J}_i$.

  - If all four jobs are small and Case (d) did not apply, then only Case (f) can apply as the sum of processing times of any two small jobs on machine M_i cannot exceed τ. Since Case (f) sets $d = 3$, at most three small jobs can be scheduled on machine M_i and the load is at most $(3/2)\tau$.
  - If three jobs are small and one job is big, then either Case (e) or Case (f) is applied by the algorithm. In Case (e), if $p_{i,j_2^{(i)}} \leq \tau/3$ then the sorting of the jobs implies that the load is at most $\tau + 2(\tau/3) = (5/3)\tau$. Then, observe that if $p_{i,j_2^{(i)}} > \tau/3$ and $\sum_{k=2}^{4} p_{i,j_k^{(i)}} \leq \tau$, then $\sum_{k=3}^{4} p_{i,j_k^{(i)}} < \tau - \tau/3 = (2/3)\tau$, and the load on machine M_i is at most $p_{i,j_1^{(i)}} + p_{i,j_3^{(i)}} + p_{i,j_4^{(i)}} \leq \tau + (2/3)\tau = (5/3)\tau$. Next if Case (f) is applied, then $p_{i,j_1^{(i)}} + p_{i,j_2^{(i)}} \leq \tau$ implies the load of machine M_i is at most $p_{i,j_1^{(i)}} + p_{i,j_2^{(i)}} + p_{i,j_3^{(i)}} \leq \tau + \tau/2 = (3/2)\tau$.
  - If two jobs are small and two jobs are big, only Case (e) applies as the sum of processing times of any two big jobs exceeds τ. Job $j_2^{(i)}$ is big, so observe that

$\sum_{k=2}^{4} p_{i,j_k^{(i)}} \leq \tau \Rightarrow \sum_{k=3}^{4} p_{i,j_k^{(i)}} < \tau - (\tau/2) = \tau/2$. Thus, the load of machine M_i is at most $\tau + \tau/2 = (3/2)\tau$.

Hence, the maximum load of a machine with $|\mathscr{J}_i| = 4$ is at most $(5/3)\tau$.

Therefore, we obtain the following results that match the inapproximability lower bounds given by Corollary 5.1.3 and Corollary 5.1.4.

Theorem 5.2.1 *There is a polynomial-time algorithm for $R||C_{max}$ when every machine can process at most two, three, or four jobs with approximation ratio 1, 3/2, or 5/3, respectively.*

Corollary 5.2.2 *There is a polynomial-time algorithm for $R||C_{max}$ restricted to job-intersection graphs that are either triangle free or diamondless with approximation ratio 1 or 3/2, respectively. Furthermore, there is a polynomial-time algorithm for $R||C_{max}$ restricted to bipartite job-intersection graphs.*

5.3 A $(2-1/(\mathscr{r}-1))$-Approximation Algorithm for Restricted Assignment with Two Job Lengths

Let $\ell_s, \ell_b \in \mathbb{Z}^+$, where $\ell_s < \ell_b$. Recall that the restricted assignment problem with two job lengths is a special case of $R||C_{max}$ where every processing time $p_{i,j} \in \{p_j, \infty\}$ and job length $p_j \in \{\ell_s, \ell_b\}$. Note that if every job has the same job length, this is equivalent to the restricted assignment problem with unit job lengths and can be solved in polynomial time [79]. So below we consider instances where the jobs have one of two distinct job lengths, and every machine can process at most $\mathscr{r} \geq 3$ jobs. By modifying the algorithm from Chapter 5.2 along with using some known results, we obtain a $(2-1/(\mathscr{r}-1))\tau$-relaxed decision algorithm. Below we assume if not all of the jobs are scheduled, the algorithm reports `FAIL`. Given estimate τ, consider the following cases in the order provided.

1. If there is a job $J_j \in J$ with no machine M_i where $p_{i,j} = p_j \leq \tau$, report `FAIL`.

2. $\ell_s > \tau/(\mathscr{r}-1)$ and $\ell_b \leq \tau$. Apply the $(2 - \ell_s/\ell_b)$-relaxed decision algorithm from Chapter 4.1 for estimate τ. If a schedule exists with makespan τ, this algorithm will compute a schedule with makespan at most

$$\Big(2 - \frac{\ell_s}{\ell_b}\Big)\tau < \left(2 - \frac{\frac{\tau}{\mathscr{r}-1}}{\tau}\right)\tau = \Big(2 - \frac{1}{\mathscr{r}-1}\Big)\tau.$$

3. $\ell_s \leq \tau/(\mathscr{r}-1)$ and $\ell_b \leq \tau/2$. Apply Theorem 2.2.1. If a schedule is produced, it has makespan at most $\tau + \max\{\ell_s, \ell_b\} \leq \tau + \tau/2 = (3/2)\tau$.

4. $\ell_s \leq \tau/(\mathscr{r}-1)$ and $\tau/2 < \ell_b \leq \tau$. Use the algorithm given in Chapter 5.2 except in Step 2, for every machine M_i proceed as follows:

   - If every job in $\mathscr{J}_i$ is small, it is possible for every job in $\mathscr{J}_i$ to be scheduled on machine M_i, so use the Machine Plan with $d = |\mathscr{J}_i|$. The load of the machine i is at most $|\mathscr{J}_i|\ell_s \leq |\mathscr{J}_i|(\tau/(\mathscr{r}-1)) \leq \mathscr{r}(\tau/(\mathscr{r}-1)) \leq (2 - 1/(\mathscr{r}-1))\tau$ as $\mathscr{r} \geq 3$.

- There is at least one big job in job set $\mathcal{J}_i$. If $\ell_b + \sum_{j\in\mathcal{S}_i} p_j \le \tau$, use the Machine Plan with $d = |\mathcal{S}_i| + 1$, where $\mathcal{S}_i$ is the set of small jobs of job set $\mathcal{J}_i$. At most one big job can be scheduled with every job in $\mathcal{S}_i$, so the load of a machine M_i is at most $\ell_b + \sum_{j\in\mathcal{S}_i} p_j \le \tau$.

  If $\ell_b + \sum_{j\in\mathcal{S}_i} p_j > \tau$, then either at most one big job can be scheduled with $|\mathcal{S}_i| - 1$ small jobs or at most all $|\mathcal{S}_i|$ small jobs are scheduled together. Use the Machine Plan with $d = \max\{|\mathcal{S}_i|, 1\}$. Since at least one job in job set $\mathcal{J}_i$ is big, $|\mathcal{S}_i| \le |\mathcal{J}_i| - 1 \le \kappa - 1$. If every job that is scheduled on machine M_i is small, then the load is at most $|\mathcal{S}_i|\ell_s \le (\kappa - 1)(\tau/(\kappa - 1)) = \tau$. Otherwise, at most one big job can be scheduled along with $|\mathcal{S}_i| - 1$ small jobs and so the load of machine M_i is at most

$$\ell_b + (|\mathcal{S}_i| - 1)\ell_s \le \tau + ((\kappa - 1) - 1)\Big(\frac{\tau}{\kappa - 1}\Big) = \Big(2 - \frac{1}{\kappa - 1}\Big)\tau.$$

Theorem 5.3.1 *There is a* $(2-1/(\kappa-1))$*-approximation algorithm for the restricted assignment problem with two job lengths when every machine can process at most* $\kappa \ge 3$ *jobs.*

5.4 Inapproximability Results for Job-Intersection Graphs with Cliques

For any instance $I = (P = (p_{i,j}), m, n)$ of $R||C_{max}$ with some $p_{i,j} = \infty$, there is another instance $I' = (P' = (p'_{i,j}), m, n)$ of $R||C_{max}$ with the same optimal solution but every $p'_{i,j} \ne \infty$: set $p'_{i,j} = p_{i,j}$ for any $p_{i,j} \ne \infty$; and if $p_{i,j} = \infty$, set $p'_{i,j}$ to some prohibitively large number, for example, $p'_{i,j} = np_{max} + 1$ where p_{max} is the largest processing time that is not ∞ in P. For $\tau \le np_{max}$, there is a schedule for instance I with makespan τ if and only if there is a schedule for instance I' with makespan τ. Every job in instance I' can be scheduled on any of the machines, so the job-intersection graph G_J for I' is the complete graph K_n. We note that an alternate construction to arrive at the complete job-intersection graph is given at the start of Section 4 in [54]. Therefore, we can carry forward the inapproximability lower bound 3/2 from the graph balancing problem with two job lengths given in Chapter 5.1.

Corollary 5.4.1 *There is no* ρ*-approximation algorithm with* $\rho < 3/2$ *for* $R||C_{max}$ *restricted to instances where the job-intersection graph is the complete graph* K_n*, unless* P = NP.

From Corollary 5.4.1, $R||C_{max}$ restricted to any superclass of the complete job-intersection graphs inherits the 3/2-inapproximability lower bound of $R||C_{max}$. We name some of these graph classes as they are of interest from a graph-theoretic standpoint. To begin, define a job-intersection graph as a *threshold graph* if it can be constructed by repeatedly performing the following two operations: insert an isolated vertex; or insert a vertex and add edges from this vertex to every other vertex presently in the graph, this vertex is called a *dominating* vertex. All complete graphs are threshold graphs, and three superclasses of threshold graphs are interval graphs, cographs, and split graphs [8, Corollary 7.1.1]. Note that all these graphs belong to the house-free graphs. A graph is called *house free* if the graph does not contain as an induced subgraph the *house graph*, shown in Figure 5.2.

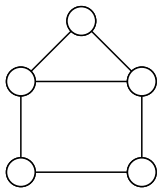

Figure 5.2: The house graph.

Corollary 5.4.2 *There is no ρ-approximation algorithm with $\rho < 3/2$ for $R||C_{max}$ restricted to instances where the job-intersection graphs belong to either the threshold graphs, interval graphs, cographs, split graphs, or house-free graphs, unless* P = NP.

Chapter 6

Scheduling Parallel Machines with a Few Bags

In this chapter we present results for makespan minimization on unrelated parallel machines with bags ($R|bag|C_{max}$) and special cases of this problem. Recall that in these scheduling problems, the jobs J are partitioned into ℓ sets called bags $B = (B_1, B_2, \ldots, B_\ell)$, and it must be that a feasible schedule must satisfy the bag constraints: no two jobs from the same bag can be scheduled on the same machine.

To begin this chapter, we provide a couple basic properties of $R|bag|C_{max}$ in Chapter 6.1. In Chapter 6.2 we give a simple ℓ-approximation algorithm for $R|bag|C_{max}$, then in Chapter 6.3 a PTAS for $R|bag|C_{max}$ when both the number of machine types and number of bags are constant is presented. In Chapter 6.4 we present a $\ell/2$-approximation algorithm for the graph balancing problem with $\ell \geq 2$ bags, this is also a polynomial-time algorithm for the graph balancing problem with $\ell = 2$ bags. In Chapter 6.5 we prove that the restricted assignment problem with bags on uniform parallel machines where the jobs all have the same length ($Q|bag, \mathcal{M}_j, p_j = 1|C_{max}$) is polynomial-time solvable, and also present a $O(m \log m)$-time algorithm for makespan minimization on identical parallel machines with bags ($P|bag|C_{max}$) when there are $\ell = 2$ bags. Finally, in Chapter 6.6, we prove that it is NP-hard to approximate with approximation ratio less than $3/2$ the restricted assignment and graph balancing problems with $\ell = 2$ and $\ell = 3$ bags, respectively, and that makespan minimization on uniform parallel machines ($Q|bag|C_{max}$) with $\ell = 2$ bags is strongly NP-hard.

6.1 A Couple Basic Properties

First, if $\ell = 1$, at most one job can be scheduled on each machine. Hence, we can solve $R|bag|C_{max}$ with one bag in polynomial time as follows: build a weighted bipartite graph $G = (J \cup M, E)$, where

$$E = \Big\{(J_j, M_i) \mid \text{job } J_j \text{ can be scheduled on machine } M_i\Big\},$$

and $w(j, i) = p_{i,j}$ for every $(J_j, M_i) \in E$. Given a weighted bipartite graph, a *maximum cardinality bottleneck matching* is a maximum (cardinality) matching $\mathcal{M}$ where the maximum length of an edge in $\mathcal{M}$ is as small as possible. Compute a maximum cardinality bottleneck

matching $\mathscr{M}$ of G [89] and for each arc $(J_j, M_i) \in \mathscr{M}$, schedule job J_j on machine M_i; there is no feasible solution if any job is not scheduled. Thus, in the sequel we focus on $R|bag|C_{max}$ when there are $\ell > 1$ bags.

Second, despite being a generalization of $R||C_{max}$, the feasible schedules of $R|bag|C_{max}$ have a strict limit of at most ℓ jobs per machine:

Property 6.1.1 *For any schedule that satisfies the bag constraints with ℓ bags, there are at most ℓ jobs scheduled on a machine.*

Proof Assume otherwise, then there exists at least one machine with more than ℓ jobs. There are ℓ bags, so if at least $\ell + 1$ jobs are scheduled on a machine, at least two jobs belong to the same bag and the bag constraints are violated. ∎

6.2 A ℓ-Approximation Algorithm for $R|bag|C_{max}$

Our approximation algorithm uses a ℓ-relaxed decision procedure. For makespan estimate τ, the idea is to treat each bag independently and simply schedule the jobs J_j in each bag $B_k \in B$ on machines M_i where $p_{i,j} \leq \tau$ so that the bag constraints are not violated. We do this by building a flow network N where there is a source s^*, a *job node* for each $J_j \in J$, a *bag-machine node* $M_{B_k,i}$ for each $M_i \in M$ and $B_k \in B$, a *machine node* for every $M_i \in M$, and a sink t^*. Do the following for each bag $B_k \in B$: for each $J_j \in B_k$, add an arc from s^* to each job node J_j and set its capacity to 1. Next, for each $J_j \in B_k$ and $M_i \in M$, if $p_{i,j} \leq \tau$, then add an arc from job node J_j to $M_{B_k,i}$ with capacity 1. For each $M_i \in M$ and $B_k \in B$, add an arc from each bag-machine node $M_{B_k,i}$ to machine node M_i with capacity 1. Finally, from each machine node $M_i \in M$, add an arc from M_i to t^* with capacity ℓ.

The ℓ-relaxed decision algorithm is as follows.

1. Build the above flow network N and compute an integral maximum flow f.
2. **For each** $J_j \in B_k$, if $f(s^*, J_j) = 0$ **then return** `FAIL`.
3. **For each** job $J_j \in B_k$, schedule J_j on machine M_i **if** $f(J_j, M_{k,i}) = 1$, and **return** this schedule.

It is not hard to see that if an integral maximum flow is computed and all the arcs incident on s^* are saturated, the jobs in bag B_k can be scheduled on the machines so as to satisfy the bag constraints, and such that each job takes at most τ time units.

Theorem 6.2.1 *There is ℓ-approximation algorithm for $R|bag|C_{max}$.*

6.3 A PTAS for $R|bag|C_{max}$ with Constant Numbers of Machine Types and Bags

Recall that two machines M_i and $M_{i'}$ have the same machine type if, for every $J_j \in J$, $p_{i,j} = p_{i',j}$. Let $N_t(\upsilon)$ be the number of machines of machine type υ, and let δ be the number of machine types. For the PTAS, a ℓ-approximate value t to the optimal makespan is computed,

so that $t/\mathscr{b} \leq OPT \leq t$; value t can be computed using the $\mathscr{b}$-approximation algorithm from Chapter 6.2.

Consider all possible schedules of some length τ. We simplify the structure of these schedules so that there is only a constant number of different machine load configurations (defined below), while increasing the length of the schedule by a factor of at most $(1+\epsilon)$ for any fixed $\epsilon > 0$. This simplification will allow us to design a PTAS, by using binary search over the interval $[t/b, t]$ to find the smallest value τ for which the algorithm given below computes a schedule

First divide the timeline of a schedule into units of length $(\tau\epsilon)/\mathscr{b}^2$ for some constant $\epsilon > 0$. The *load configuration* of machine M_i is a vector $(\ell_{i,1}, \ell_{i,2}, \ldots, \ell_{i,\mathscr{b}})$ such that if job J_j from bag B_k is scheduled on M_i then $(\ell_{i,k}-1)(\tau\epsilon)/\mathscr{b}^2 < p_{i,j} \leq \ell_{i,k} \cdot (\tau\epsilon)/\mathscr{b}^2$. The maximum number of values each $\ell_{i,k}$ can take is $1 + \lceil \frac{\tau}{\frac{\tau\epsilon}{\mathscr{b}^2}} \rceil = 1 + \lceil \frac{\mathscr{b}^2}{\epsilon} \rceil$, so $\ell_{i,k} \in \{0, 1, 2, \ldots, \lceil \mathscr{b}^2/\epsilon \rceil\}$. As there are $\mathscr{b}$ values in any load configuration, the number of possible load configurations is then $(1 + \lceil \mathscr{b}^2/\epsilon \rceil)^b =: D$, a constant; index these load configurations $1, 2, \ldots, D$. Let vector $(c_{1,1}, c_{1,2}, \ldots, c_{1,D}, c_{2,1}, c_{2,2}, \ldots, c_{2,D}, \ldots, c_{\delta,1}, c_{\delta,2}, \ldots, c_{\delta,D})$ be a *schedule configuration*, where $c_{\upsilon,\mu}$ is the number of machines with machine type υ that have load configuration μ. There are m machines, so each $c_{\upsilon,\mu} \in \{0, 1, \ldots, m\}$ and because there are δD many elements in a schedule configuration, the total number of possible schedule configurations is $O(m^{\delta D})$, a polynomial function as δ and D are constant. A schedule configuration is *valid* if $\sum_{\upsilon=1}^{\delta} \sum_{\mu=1}^{D} c_{\upsilon,\mu} = m$, and $\sum_{\mu=1}^{D} c_{\upsilon,\mu} = N_t(\upsilon)$, for $\upsilon = 1, 2, \ldots, \delta$. For each valid schedule configuration there are exactly m load configurations, one for each machine. That is, for each $c_{\upsilon,\mu} > 0$, assign load configuration μ to $c_{\upsilon,\mu}$ machines of machine type υ. Then, the makespan of a valid schedule configuration is $\max_{1 \leq i \leq m} \left\{ \sum_{k=1}^{b} \ell_{i,k} (\frac{\tau\epsilon}{\mathscr{b}^2}) \right\}$.

We compute all valid schedule configurations for makespan τ and choose one for which the jobs can be allocated to the machines according to that schedule configuration. If there is a feasible schedule, at least one such schedule configuration exists where for each $J_j \in J$ with $J_j \in B_k$, there is a machine M_i with $p_{i,j} \leq \ell_{i,k}(\frac{\tau\epsilon}{\mathscr{b}^2}) \leq \lceil \frac{\mathscr{b}^2}{\epsilon} \rceil (\frac{\tau\epsilon}{\mathscr{b}^2})$ where $\frac{\mathscr{b}^2}{\epsilon}(\frac{\tau\epsilon}{\mathscr{b}^2}) = \tau \leq \lceil \frac{\mathscr{b}^2}{\epsilon} \rceil (\frac{\tau\epsilon}{\mathscr{b}^2})$. To find this schedule we proceed as follows. For each valid schedule configuration, assign to machine M_i a load configuration L_i as described above. Then consider each bag B_k, $k = 1, 2, \ldots, \mathscr{b}$, and build a bipartite graph $G_k = (B_k \cup M, E_k)$, where $E_k = \left\{ (j, M_i) \mid M_i \in M,\ j \in B_k, (\ell_{i,k}-1)\left(\frac{\tau\epsilon}{\mathscr{b}^2}\right) < p_{i,j} \leq \ell_{i,k}\left(\frac{\tau\epsilon}{\mathscr{b}^2}\right) \right\}$. Compute a maximum matching of G_k, and for each arc (J_j, M_i) in the matching, schedule J_j on M_i. Discard the schedule if at least one job of bag B_k is not scheduled. Otherwise, for $k = 1, 2, \ldots, \mathscr{b}$, a matching of size $|B_k|$ is computed for G_k and thus every job $J_j \in B_k$ is scheduled. This will assign at most one job from each bag B_k to each machine, so a feasible schedule is produced.

Let machine M_λ be a machine that finishes last in the schedule configuration with minimum makespan τ^* selected by the algorithm and let $L^*_\lambda = (\ell^*_{\lambda,1}, \ell^*_{\lambda,1}, \ldots, \ell^*_{\lambda,b})$ be its load configuration. Note that for each job $J_j \in B_k$ on M_λ, $p_{\lambda,j} \leq \ell^*_{\lambda,k}(\tau^*\epsilon)/\mathscr{b}^2$, but $p_{\lambda,j} > (\ell^*_{\lambda,k}-1)(\tau^*\epsilon)/\mathscr{b}^2$ as otherwise there would be another schedule configuration of lesser makespan where all the jobs can be allocated to the machines. Since $\tau^* \geq OPT$,

$$\sum_{\substack{\text{job } J_j \text{ scheduled} \\ \text{on machine } M_\lambda}} p_{\lambda,j} \geq OPT > \sum_{k=1}^{\mathscr{b}} \max\left\{ (\ell^*_{\lambda,k} - 1)\frac{\tau^*\epsilon}{\mathscr{b}^2}, 0 \right\}.$$

Therefore,

$$\sum_{\substack{\text{job } J_j \text{ scheduled} \\ \text{on machine } M_\lambda}} p_{\lambda,j} \leq \sum_{k=1}^{\mathscr{b}} \ell^*_{\lambda,k} \frac{\tau^* \epsilon}{\mathscr{b}^2} \leq \sum_{k=1}^{\mathscr{b}} \max\left\{(\ell^*_{\lambda,k} - 1)\frac{\tau^* \epsilon}{\mathscr{b}^2}, 0\right\} + \sum_{k=1}^{\mathscr{b}} \frac{\tau^* \epsilon}{\mathscr{b}^2}$$

$$< OPT + \sum_{k=1}^{\mathscr{b}} \frac{\tau^* \epsilon}{\mathscr{b}^2},$$

and since $t/\mathscr{b} \leq OPT \leq \tau^* \leq t$, the makespan is at most

$$OPT + \sum_{k=1}^{\mathscr{b}} \frac{\tau^* \epsilon}{\mathscr{b}^2} = OPT + \left(\frac{\tau^*}{\mathscr{b}}\right)\epsilon \leq OPT + \left(\frac{t}{\mathscr{b}}\right)\epsilon \leq (1+\epsilon)OPT.$$

Theorem 6.3.1 *There is a PTAS for $R|bag|C_{max}$ with machine types when both the number $\mathscr{b}$ of bags and the number δ of machine types are constant.*

Consider makespan minimization on uniform machines with bags ($Q|bag|C_{max}$). The processing time for a job on a machine depends on the speed of the machine. Therefore, the number of machine types is in fact the number of machine speeds.

Corollary 6.3.2 *There is a PTAS for $Q|bag|C_{max}$ when both the number of distinct machine speeds and the number of bags are constant.*

6.4 A $\mathscr{b}/2$-Approximation Algorithm for the Graph Balancing Problem with $\mathscr{b} \geq 2$ Bags

Recall that in the graph balancing problem with bags the jobs and machines can be represented as a weighted multigraph $G = (V, E)$, where the jobs are edges i.e. $E = \bigcup_{k=1}^{b} B_k$, each edge $e \in E$ has length $p_e \in \mathbb{Z}^+$, and the machines are the vertices. We remark that in the case with bags, we must assume the multigraph may contain self-loops (and does not have dedicated loads) as self-loops might belong to different bags. Here we continue to use m and n to be the number of machines and jobs, respectively. Let $G_{B_k} = (V_{B_k}, B_k)$ where vertex $v \in V_{B_k}$ if $v \in e \in B_k$. We call a maximally connected component of G_{B_k} a *bag component*. Recall that a *pseudoforest* is a collection of trees and unicyclic graphs (, i.e. 1-trees), a *unicyclic graph* is a connected graph that contains exactly one cycle.

Property 6.4.1 *Consider the graph balancing problem with bags. If there is a schedule S that satisfies the bag constraints, then for every $B_k \in B$, G_{B_k} is a pseudoforest.*

Proof Assume this is not the case, then there is a bag component C_{B_k} of G_{B_k} that contains at least two cycles. Observe that then there are more edges than vertices, thus in S at least two edges in C_{B_k} must be directed toward the same vertex, violating the bag constraints. ∎

In the sequel we assume that the input multigraph G satisfies the conditions of Property 6.4.1, this allows us to bound the number of possible feasible orientations for the edges in a bag component. A bag component that is a tree $T = (V_T, E_T)$ has at most $|V_T|$ possible

orientations: select each vertex as the root of the tree and direct all edges away from it. In addition, there are at most two possible orientations in a unicyclic bag component: select one of two directions to direct all the edges in the cycle, and then direct all other edges away from the cycle. We use these facts in our algorithm.

Our algorithm employs a $\mathscr{6}/2$-relaxed decision procedure. Let $l_L(u)$ be the load contributed by the edge with the largest edge length directed toward vertex u in G; hence $l_L(u) = 0$ if no edge is directed toward u. We note that if $l_L(u) > \tau/2$ then no other edges with length larger than $\tau/2$ can be directed toward u without the makespan exceeding τ; we call an edge e a *big* edge when its length $p_e > \tau/2$ and an edge is *small* if $p_e \leq \tau/2$.

The $\mathscr{6}/2$-relaxed decision algorithm uses a set D to store the edges that have already been assigned a direction. Initially $D = \emptyset$ and if a schedule exists, at the end D will contain all the edges in G. First, if any edge $e \in E$ has length larger than τ return `FAIL`. While there is an edge in $E \setminus D$ do the following. Compute a bag component C of $G \setminus D$ and perform an *expansion* of C by using the procedure described in Step (2) of the algorithm below. For each feasible orientation of the edges in C an expansion forces edges away from C if this does not violate the bag constraints and $l_L(u) + p_e \leq \tau$ for every $u \in C$ and edge e incident on u. The forcing of edges away "expands" C and this process will continue "expanding" C until no more edges need to be forced in a certain direction or infeasibility is determined. If an expansion is successfully computed, directions for a set C_E of edges is found and so we set $D = D \cup C_E$. The process is then repeated if there are any undirected edges left in $E \setminus D$. Otherwise, if no expansion was found another orientation for C is considered and another expansion is computed. The algorithm returns `FAIL` if there are no more orientations to try. We assume below that each $l_L(u)$ is updated as the direction of edges are changed. Now we formally describe the algorithm.

1. Set $D = \emptyset$. If any edge $e \in E$ has length $p_e > \tau$, return `FAIL`.
2. **While** $E \setminus D$ is not empty:
   (a) Compute a bag component C of $G \setminus D$.
   (b) Find a new orientation of the edges in C for which at most one edge from each bag is directed to the same vertex and any two edges e, e' directed to the same vertex satisfy $p_e + p_{e'} \leq \tau$. If there are no more new orientations to try for C **return** `FAIL`. Let C' be the set of vertices u where an edge is directed toward u by this step.
   (c) **While** there is a vertex $u \in C'$ and undirected edge $e = \{u, v\}$ in $E \setminus D$ where $l_L(u) + p_e > \tau$:
      i. Direct e from u to v; then direct all edges of the same bag as e that are reachable from v away from u. Add to C' all vertices whose loads increased in this step.
      ii. **If** any vertex $w \in C'$ has two edges from the same bag directed toward it or **if** there are two edges e and e' directed toward w so that $p_e + p_{e'} > \tau$ **then** reset all loads and edges directed by this iteration of Step (2) and go to Step (2b).
   (d) Let C_E be the set of edges that were assigned a direction in Steps (2b) and (2c). Set $D = D \cup C_E$.
3. **Return** schedule corresponding to the orientation of the edges.

The time complexity of this algorithm is $O(n^2m + m^2n)$. Let $C_1, C_2, \ldots, C_h$ be the bag components selected by the algorithm in Step (2a) in the order they were chosen. If after expanding a bag component C_i a vertex u has at least one edge directed to it, we say that u has been *expanded*.

Lemma 6.4.2 *If the expansion of C_h is attempted by the algorithm and it returns* `FAIL`*, then there is no schedule with makespan at most τ.*

Proof If the algorithm returns `FAIL` after attempting to expand C_h, then for every possible orientation of C_h there is a vertex $w \in C'$ where either: two edges from the same bag are directed toward w, violating the bag constraints; or two edges e, e' with $p_e + p_{e'} > \tau$ have been directed toward w. Note that after expanding any of the bag components $C_1, C_2, \ldots, C_{h-1}$ for every expanded vertex u the following properties hold: at most one edge from each bag is directed to u; any two edges directed to u have combined length at most d; and any undirected edge $e = \{u, v\}$ satisfies $l_L(u) + p_e \leq \tau$.

These properties allow us to deal with each bag component C_i independently from the previously expanded bag components $C_1, \ldots, C_{i-1}$. In particular, if the algorithm returns `FAIL` when trying to expand C_h, then there cannot be a schedule of length at most τ. To see this assume the opposite, that there is a schedule S of makespan at most τ. Since no vertex expanded by the expansion of $C_1, C_2, \ldots, C_{h-1}$ is incident to an undirected edge from the same bag as C_h, then the algorithm will consider the same orientation for the edges of C_h as the orientation chosen by S. Since the makespan of S is at most τ, if there is a vertex u in C_h incident to an undirected edge $e = \{u, v\}$ for which $l_L(u) + p_e > \tau$ then S orients e from u to v and any undirected edge from the same bag as e is directed away from u. Note that this is exactly what our algorithm does. Any further vertices u expanded by this process incident on undirected edges are handled in the same manner. Since our algorithm assigns directions to the edges exactly as S, then if S exists our algorithm must successfully expand C_h. ∎

Lemma 6.4.3 *If the algorithm produces a schedule, the makespan of the schedule is at most $(\ell/2)\tau$.*

Proof Consider the load of any vertex $w \in V$ with $l_L(w) > 0$. There are ℓ bags, so at most ℓ different edges $e_1, e_2, \ldots, e_\ell$ are directed toward w, where $e_1 \in B_1, e_2 \in B_2, \ldots, e_\ell \in B_\ell$. If $l_L(w) \leq \tau/2$, then all edges oriented toward w are small and the load of w is at most $\ell\tau/2$. Otherwise $l_L(w) > \tau/2$; since the algorithm only proceeds to Step (2) if every edge $e \in E$ has weight $p_e \leq \tau$, then $\tau/2 < l_L(w) \leq \tau$. The first time a big edge is directed toward w causing $l_L(w) > \tau/2$ all undirected big edges incident on w are directed away from w; so besides this big edge all other edges directed to w are small. By the condition of the while loop $l_L(w) + p_e \leq \tau$ for every small edge directed to w and so $p_e \leq \tau - l_L(w)$. Thus the load of w is at most

$$\sum_{k=1}^{\ell} p_{e_k} \leq l_L(w) + (\ell - 1)(\tau - l_L(w)) = \tau + (\ell - 2)(\tau - l_L(w)) \leq \tau + \frac{(\ell - 2)\tau}{2} = \frac{\ell\tau}{2},$$

where the last inequality holds because $l_L(w) > \tau/2$. Therefore, the makespan of a schedule produced by the algorithm is at most $(\ell/2)\tau$. ∎

Theorem 6.4.4 *There is a $\mathscr{b}/2$-approximation algorithm for the graph balancing problem with $\mathscr{b} \geq 2$ bags.*

Corollary 6.4.5 *There is a polynomial-time algorithm for the graph balancing problem with two bags.*

6.5 Polynomial-time Solvable Problems

6.5.1 A Polynomial-Time Algorithm for $Q|bag, \mathscr{M}_j, p_j = 1|C_{max}$

In this section we consider the restricted assignment problem on uniform parallel machines with bags where every job has the same length. Without loss of generality we can assume that all the machines have speeds $s_1, \ldots, s_m \in \mathbb{Z}^+$. Let the least common multiple of the speeds $s_1, \ldots, s_m$ be c. The above problem is equivalent to when every job $J_j \in J$ has length $p_j = c$, so for convenience we assume below that every job has length $p_j = c$. Observe that since c is the least common multiple of the speeds, p_j/s_i is integral for all $J_j \in J$ and $M_i \in M$. We employ a 1-relaxed decision procedure. Note that in the special case when there is one bag for each job, our algorithm is exactly the algorithm of Lin and Li [79] for $Q|\mathscr{M}_j, p_j = 1|C_{max}$.

Let a *conflict machine set for bag* B_k be $\mathscr{C}(B_k) = \{M_i \in M \mid \exists J_j, J_{j'} \in B_k : M_i \in \mathscr{M}_j \cap \mathscr{M}_{j'}\}$, where $\mathscr{M}_j$ and $\mathscr{M}_{j'}$ are the sets of machines where for jobs J_j and $J_{j'}$ can be scheduled, respectively. As a result of this definition, if there is a machine $M_i \notin \mathscr{C}(B_k)$, then at most one job in B_k can be scheduled on machine M_i. In our algorithm we first build a flow network N with a source s^* and sink t^* as follows. First, there will be a *job node* for each job $J_j \in J$; then for each $k = 1, \ldots, \mathscr{b}$, create a *conflict machine node* for every machine $M'_i \in \mathscr{C}(B_k)$, and a *machine node* for each machine $M_i \in M$. To avoid ambiguity, we write M'_i whenever we refer to a conflict machine node of a machine M_i, and M_i when we refer to the machine node for machine M_i. We add arcs as follows: add arcs with capacity 1 from the source to each job node; if job $J_j \in B_k$ can be scheduled on machine M_i: (i) if $M_i \in \mathscr{C}(B_k)$ then add an arc from J_j to the machine conflict node M'_i of bag B_k with capacity 1, (ii) otherwise add an arc from J_j to machine node M_i with capacity 1. Add an arc with flow capacity 1 from each machine conflict node M'_i to its corresponding machine node M_i and include an arc from each machine node M_i to the sink with capacity $\lfloor (s_i\tau)/c \rfloor$. See Figure 6.1 for an example network.

Lemma 6.5.1 *There is an integral flow f that saturates all the arcs incident on s^* if and only if there is a schedule with makespan at most τ such that the bag constraints are satisfied.*

Proof First, let us show that if there is a flow f for N that saturates all the arcs incident on the source, then there a schedule with makespan at most τ and the bag constraints are satisfied. As all the arcs incident on the source are saturated by f, one unit of flow is sent from the source to each job node. The flow from each job node J_j is either sent directly to a machine node M_i or a machine conflict node M'_i. If the unit of flow is sent to M_i or to M'_i, schedule job J_j on machine M_i.

Now we must show that by employing the above process, we obtain a schedule such that the bag constraints are satisfied and the makespan of the schedule is at most τ. All the arcs

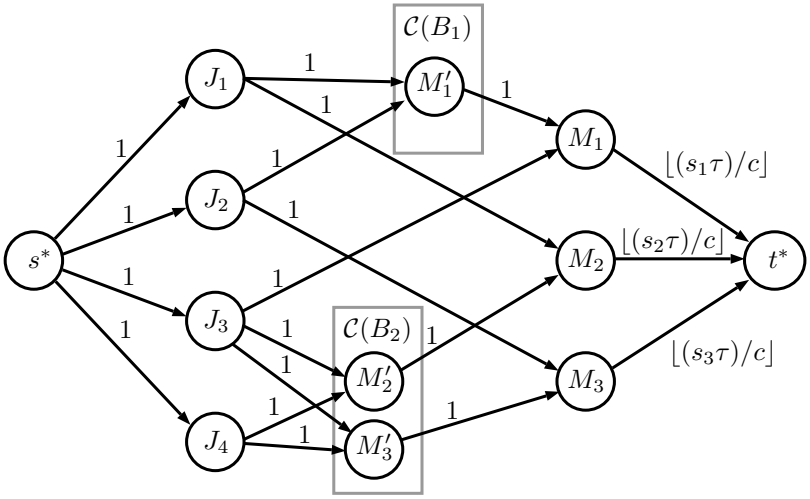

Figure 6.1: Flow network N for the following instance: $n = 4$ jobs $J = \{J_1, J_2, J_3, J_4\}$ and $m = 3$ machines $M = \{M_1, M_2, M_3\}$, the eligibility constraints for the jobs are $\mathscr{M}_1 = \{M_1, M_2\}$, $\mathscr{M}_2 = \{M_1, M_3\}$, $\mathscr{M}_3 = \{M_1, M_2, M_3\}$, and $\mathscr{M}_4 = \{M_2, M_3\}$, and $\mathscr{b} = 2$ bags where $B_1 = \{J_1, J_2\}$, and $B_2 = \{J_3, J_4\}$.

incident on the source are saturated in f, and so $|f| = n$. All the arcs except those incident on the sink have capacity 1, so by flow conservation the n units of flow from the job nodes are received by the machine nodes. We schedule job J_j on a machine if its corresponding machine node receives 1 unit of flow from job node J_j, thus, the schedule must contain all n jobs. For any machine node M_i,

$$\left(\sum_{J_j \in J} f(J_j, M_i) + \sum_{M'_i \in \mathscr{C}(B_k) | k=1,\ldots,\mathscr{b}} f(M'_i, M_i) \right) \leq c(M_i, t^*) = \left\lfloor \frac{s_i \tau}{c} \right\rfloor,$$

so no more than $\lfloor (s_i\tau)/c \rfloor$ jobs can be assigned to machine M_i. Every job has length c, so the load of machine M_i cannot exceed $(c/s_i)\lfloor (s_i\tau)/c \rfloor \leq \tau$. Machines where at least two jobs from B_k can be scheduled are in set $\mathscr{C}(B_k)$, and all jobs in $J_j \in B_k$ that can be scheduled on a machine M_i in $\mathscr{C}(B_k)$ have an arc from job node J_j to conflict machine node M'_i. Each arc leaving a conflict machine node has capacity 1, so at most one of the jobs from a bag B_k can be scheduled to a machine, and so the bag constraints are satisfied.

Next, we show that if there is a schedule with makespan at most τ such that the bag constraints are satisfied, then we can build a flow function f such that all the arcs incident on the source are saturated. As there is a job for each job node, set $f(s^*, J_j) = 1$. For each job J_j scheduled on a machine M_i where $J_j \in B_k$ for some bag B_k, if machine M_i is not in $\mathscr{C}(B_k)$ set $f(J_j, M_i) = 1$, otherwise set $f(J_j, M'_i) = 1$. At most one job in bag B_k is scheduled on each machine $M_i \in \mathscr{C}(B_k)$ and at most one unit of flow is received at machine conflict node M'_i of B_k, so this unit of flow can be sent to machine node M_i as $c(M'_i, M_i) = 1$. A unit of flow is sent from each job node J_j to the machine node M_i where J_j is scheduled, and the sum of the flows entering machine node M_i is equal to the number of jobs scheduled on machine M_i. Each job takes c/s_i time units on machine M_i and the makespan is at most τ, so at most $\lfloor (s_i\tau)/c \rfloor$ jobs can be scheduled on machine M_i. This implies that at most $\lfloor (s_i\tau)/c \rfloor$ units of flow are received

by machine node M_i and $f(M_i, t) \leq c(M_i, t^*) = \lfloor (s_i\tau)/c \rfloor$, therefore we can send the flow from every machine node M_i to the sink t^*. ∎

The 1-relaxed decision algorithm is the following: build the flow network N described above and compute an integral maximum flow f; if f does not saturate at least one arc incident on the source, return `FAIL`; otherwise all the arcs incident on the source are saturated, and for each job $J_j \in J$, schedule job J_j on machine M_i if there is flow sent from job node J_j to machine node M_i.

Theorem 6.5.2 *There is a polynomial-time algorithm for $Q|bag, \mathscr{M}_j, p_j = 1|C_{max}$.*

6.5.2 A $O(m \log m)$-Time Algorithm for $P|bag|C_{max}$ with $\ell = 2$ Bags

If $m < \max\{|B_1|, |B_2|\}$ then two jobs of the same bag must go on the same machine and no solution exists, so we assume that $m \geq \max\{|B_1|, |B_2|\}$. Introduce dummy jobs so that there are $2m$ jobs where $|B_1| = m$ and $|B_2| = m$; each dummy job has length zero. Then, our problem is equivalent to the following problem: given two sequences $A = a_1, a_2, \ldots, a_m$ and $A' = a'_1, a'_2, \ldots, a'_m$ representing the job lengths, find m disjoint pairs where each pair consists of one element from A and the other from A' such that the maximum sum of a pair is minimized. It is trivial to solve this problem: sort the numbers in both sequences so that $a_1 \geq a_2 \geq \cdots \geq a_m$ and $a'_1 \leq a'_2 \leq \cdots \leq a'_m$, then the m pairs are $(a_1, a'_1), (a_2, a'_2), \ldots, (a_m, a'_m)$. Then we can produce a schedule: for each pair (a_i, a'_i), $i = 1, 2, \ldots, m$, schedule the non-dummy jobs of pair i on M_i. Since $m \geq \max\{|B_1|, |B_2|\}$, this algorithm takes $O(m \log m)$ time.

Theorem 6.5.3 *$P|bag|C_{max}$ with $\ell = 2$ bags is solvable in $O(m \log m)$ time.*

6.6 Inapproximability and Complexity

6.6.1 Restricted Assignment Problem with $\ell = 2$ Bags and Two Job Lengths

To begin, we prove that restricted assignment problem with $\ell = 2$ bags where the job lengths are either 1 or 2 has no approximation algorithm with approximation ratio less than $3/2$, unless $\mathsf{P} = \mathsf{NP}$ (Corollary 6.6.2). To do this we reduce from the 3-dimensional matching problem ([SP1] in [29]). The result follows from Theorem 6.6.1 below. We note that our reduction is similar to the one given by Lenstra *et al.* [72], but their reduction assumes there are $\ell = n$ bags.

Problem: 3-Dimensional Matching (3DM)

Input: Three disjoint sets $X = \{x_1, x_2, \ldots, x_{m'}\}, Y = \{y_1, \ldots, y_{m'}\}, Z = \{z_1, \ldots, z_{m'}\}$, and a set $T \subseteq X \times Y \times Z$ of triples.

Output: Is there a set $T' \subseteq T$ containing m' triples, such that for any pair of triples $(x_k, y_k, z_k), (x_\ell, y_\ell, z_\ell) \in T'$, $x_k \neq x_\ell$, $y_k \neq y_\ell$, and $z_k \neq z_\ell$?

Theorem 6.6.1 *It is NP-hard to decide whether there is a schedule with makespan at most 2 for the restricted assignment problem with $\ell = 2$ bags when the jobs lengths $p_j \in \{1, 2\}$.*

Proof We give a reduction from the 3-dimensional matching problem. Build a scheduling instance as follows. For each triple $t \in T$ where element $z \in t$ and $z \in Z$, create a machine M_t of *type* z. Next, each element in $X \cup Y$ is a job j, where we place j in bag B_1 if $j \in X$ and in bag B_2 if $j \in Y$; j has processing time $p_{t,j} = 1$ on machine M_t if $j \in t$, and $p_{t,j} = \infty$ otherwise. Let $deg(z)$ be the number of triples of T that contain element $z \in Z$. Then for each element $z \in Z$, create $(deg(z) - 1)$ *dummy* jobs of *type* z, where each dummy job j takes 2 time units on machines of type z, and $p_{t,j} = \infty$ otherwise. Place all the dummy jobs in bag B_1.

We show there is a schedule with makespan at most 2 that satisfies the bag constraints if and only if there is a 3DM T' of size m'.

($\Rightarrow$) Since the makespan is at most 2, the dummy jobs are scheduled on $\sum_{z \in Z} (deg(z) - 1)$ of the machines. As there are $deg(z)$ machines of type $z \in Z$ and $(deg(z) - 1)$ of these machines have scheduled dummy jobs in them, there are exactly m' machines scheduled with the $2m'$ non-dummy jobs, one machine for each type. Notice that if any one of these m' machines schedules exactly one non-dummy job, then either three non-dummy jobs are scheduled together on a machine or a non-dummy job is scheduled with a dummy job, but the schedule then has makespan at least 3 if either is the case. This implies the non-dummy jobs, are scheduled two jobs per machine on the remaining m' machines. By the bag constraints, each pair of non-dummy jobs on these machines has one job from bag $B_1 = X$ and the other job from $B_2 = Y$.

Constructing the 3DM T' is simple. For each pair of non-dummy jobs $x \in X$ and $y \in Y$ scheduled on a machine of type $z \in Z$, include the triple (x, y, z) in T'. There is only one job per element in $X \cup Y$ and exactly one triple is selected for each $z \in Z$, therefore T' has m' pairwise disjoint triples.

($\Leftarrow$) For each triple $t = (x, y, z) \in T'$, schedule jobs x and y on machine M_t, then uniquely schedule all the dummy jobs of type z on the remaining $(deg(z) - 1)$ machines of type z. Either a machine has a dummy job scheduled on it or two non-dummy jobs scheduled on it, so the makespan of this schedule is 2. Each $x \in X = B_1$ and $y \in Y = B_2$, so machines with non-dummy jobs respect the bag constraints. On the remaining machines, exactly one dummy job is scheduled so, again, the bag constraints are satisfied. Therefore, the schedule constructed has makespan at most 2 and satisfies the bag constraints. ∎

Corollary 6.6.2 *There is no ρ-approximation algorithm with $\rho < 3/2$ for the restricted assignment problem with $\mathscr{b} = 2$ bags where the job lengths are either 1 or 2, unless* P = NP.

6.6.2 Graph Balancing Problem with $\mathscr{b} = 3$ Bags and Two Job Lengths

In this section we show that when there are $\mathscr{b} \geq 3$ bags, it is NP-hard to approximate the graph balancing problem with $\mathscr{b}$ bags with approximation ratio less than $3/2$; in the graph balancing problem with bags edges are assigned to bags. To do this we extend the reduction in Chapter 2.2.3. As stated in Chapter 6.4, for the graph balancing problem with bags we presume that there are self-loops instead of dedicated loads, so we will state any non-zero dedicated load in the original reduction as a self-loop. That is, when the dedicated load is set to $3 - |y_i|$ for each clause vertex y_i in Chapter 2.2.3, add instead a self-loop of length $3 - |y_i|$ on clause vertex y_i if $3 - |y_i| > 0$, where $|y_i|$ is the number of literals in clause y_i.

Now we describe our extension to this reduction that will assign each edge to a bag. Create a modified version of G called G', where, for each self loop incident on a clause vertex y_i in G, replace the self-loop with a new vertex y_i' and *self edge* $\{y_i, y_i'\}$ in G'; each self-edge in G' corresponds to a self-loop in G. Note that G' is simple.

Lemma 6.6.3 *There is an edge colouring of G' that uses at most four colours, this colouring can be computed in polynomial time.*

Proof Let the four colours be $\eta_1, \eta_2, \eta_3, \eta_4$. Begin by assigning the colour η_4 to all tautologous edges, then consider the subgraph G'' of G' consisting of the same vertices but only the uncoloured edges. Observe that every edge in G'' either has a literal vertex and a clause vertex as its endpoints or is a self-edge with one endpoint that is a leaf, thus G'' is bipartite. Since G'' is bipartite and the maximum degree of any vertex in G'' is three, by Vizing's Theorem [98] there is an edge colouring of G'' using three colours η_1, η_2, η_3, this edge colouring can be computed in polynomial time. ∎

Using Lemma 6.6.3 we can assign the edges in G to three bags: the edges coloured in G' using colour η_1 are in bag B_1, the edges coloured in G' using colour η_2 are in bag B_2, and the edges in G' coloured with η_3 are in bag B_3. Finally, place the edges coloured with η_4 in any of the three bags.

If a schedule has makespan at most 2, observe that the edges coloured in G' with η_4 are the tautologous edges, each of these edges has weight 2 and if each tautologous edge is directed towards a literal vertex, all other edges incident on such a literal vertex cannot be directed toward it. All the edges incident on literal vertices other than the tautologous edges cannot violate the bag constraints at literal vertices as they were given different colours, hence are assigned to different bags. As a result, no tautologous edge can violate the bag constraints for any schedule with makespan at most 2. All other edges in G have the property that no two edges incident on any clause vertex in G' share the same colour, so this ensures that any schedule computed does not violate the bag constraints at the clause vertices. By Theorem 2.2.2, G has an orientation with makespan at most 2 if and only if ϕ is satisfied. Therefore, by assigning the bags as we have described, the reduction carries forward.

Theorem 6.6.4 *There is no ρ-approximation algorithm with $\rho < 3/2$ for the graph balancing problem with $\ell \geq 3$ bags where job lengths are either 1 or 2, unless* P = NP.

6.6.3 $Q|bag|C_{max}$ with $\ell = 2$ Bags

For $P|bag|C_{max}$ with $\ell = 3$ bags and $Q|bag|C_{max}$ with $\ell = 2$ bags, we show that both are strongly NP-hard. We reduce from numerical 3-dimensional matching ([SP16] in [29]), which is known to be NP-complete in the strong sense.

Problem: Numerical 3-Dimensional Matching

Input: 3 disjoint sets $X = \{a_1, a_2, \ldots, a_{m'}\}$, $Y = \{a_{m'+1}, \ldots, a_{2m'}\}$, $Z = \{a_{2m'+1}, \ldots, a_{3m'}\}$, and a value $\beta \in \mathbb{Z}^+$; every element $a_j \in X \cup Y \cup Z$ has a size $s(a_j) \in \mathbb{Z}^+$.

Output: Are there disjoint triples $A_1, \ldots, A_{m'}$ where each triple A_i contains exactly one element of X, one element of Y, and one element of Z, such that $\sum_{a_j \in A_i} s(a_j) = \beta$?

Notice that if an instance of $P|bag|C_{max}$ with $\mathscr{b} = 3$ bags has exactly $3m$ jobs, Property 6.1.1 implies that every machine in a feasible schedule processes three jobs. By exploiting the fact that the bags are disjoint and that we cannot place any two jobs from the same bag together on a machine, we obtain a straightforward reduction from numerical 3-dimensional matching to $P|bag|C_{max}$ with $\mathscr{b} = 3$ bags. This reduction was independently presented by Dokka *et al.* [19].

Theorem 6.6.5 (Dokka *et al.* [19]) *$P|bag|C_{max}$ with $\mathscr{b} = 3$ bags is strongly* NP*-hard.*

Proof To prove the claim we consider the decision variant of $P|bag|C_{max}$ and show it is NP-complete in the strong sense. Clearly the decision variant of $P|bag|C_{max}$ is in NP. We show a reduction from numerical 3-dimensional matching to $P|bag|C_{max}$ with $\mathscr{b} = 3$ bags. Without loss of generality we can assume that $s(a_j) \leq \beta$ for all $a_j \in X \cup Y \cup Z$ and $\sum_{a_j \in X \cup Y \cup Z} s(a_j) = m'\beta$. Let the number of machines $m = m'$, the number of jobs $n = 3m'$, where each job j is an element $a_j \in X \cup Y \cup Z$ with length $p_j = s(a_j)$, and bags $B_1 = X$, $B_2 = Y$, and $B_3 = Z$.

The above reduction can be performed in polynomial time. We now show that there is a schedule with makespan β that satisfies the bag constraints if and only if there are disjoint triples $A_1, \ldots, A_{m'}$ where $\sum_{a_j \in A_i} s(a_j) = \beta$, for each $i = 1, \ldots, m'$.

($\Rightarrow$) Since there are $m = m'$ machines and $n = 3m'$ jobs, by Observation 6.1.1 in a schedule that respects the bag constraints every machine has exactly three jobs scheduled on it. Put every job scheduled on machine M_i in triple A_i, $i = 1, \ldots, m$.

As the bag constraints are satisfied, no two jobs from the same bag are placed on machine M_i and so exactly one element of X, one of Y, and one of Z are in each triple A_i. Each job $J_j \in J$ is an element in $X \cup Y \cup Z$, thus $\sum_{J_j \in J} p_j = \sum_{a_j \in X \cup Y \cup Z} s(a_j)$. To show that the sum of the sizes of the elements in a triple A_i is β, assume there is a schedule with makespan β but there is at least one machine with load less than β. Then

$$\sum_{J_j \in J} p_j = \sum_{i=1}^{m} \left(\sum_{\substack{\text{job } J_j \text{ scheduled} \\ \text{on machine } M_i}} p_j \right) < m\beta = m'\beta = \sum_{a_j \in X \cup Y \cup Z} s(a_j),$$

which implies such a schedule cannot exist, so every machine has load β and each triple has elements of total size β.

($\Leftarrow$) For triples $A_1, \ldots, A_{m'}$, schedule each job J_j with length $p_j = s(a_j)$ for each $a_j \in A_i$ on machine M_i. Since there are $m = m'$ machines, every job is scheduled. For each triple A_i, $\sum_{a_j \in A_i} s(a_j) = \beta$ is satisfied so the makespan of our schedule is β. Since we set the bags $B_1 = X$, $B_2 = Y$, and $B_3 = Z$, each machine has exactly one job from each bag and the bag constraints are satisfied. ∎

When the machines are uniform the problem is NP-hard with only two bags.

Theorem 6.6.6 *$Q|bag|C_{max}$ with $\mathscr{b} = 2$ bags is strongly* NP*-hard.*

Proof Consider the decision variant of $Q|bag|C_{max}$ with $\mathscr{b} = 2$ bags. Clearly this decision problem is in NP. Again, we reduce from numerical 3-dimensional matching. We can assume without loss of generality that for every element $s(a_j) \leq \beta$, $a_j \in X \cup Y \cup Z$, and $\sum_{a_j \in X \cup Y \cup Y} s(a_j) =$

$m'\beta$. Let the number of machines $m = m'$, the number of jobs $n = 2m'$ where job $J_j \in J$ corresponds to element $a_j \in X \cup Y$ and has length $p_j = s(a_j)$, and the bags are $B_1 = X$ and $B_2 = Y$. We associate each machine M_i with a unique element $z_i \in Z$ and set the speed of M_i to $s_i = (\beta - s(z_i))/\beta$. Clearly this reduction can be done in polynomial time.

We show that there is a schedule with makespan at most β if and only if there are disjoint triples $A_1, \ldots, A_{m'}$, where $\sum_{a_j \in A_i} s(a_j) = \beta$, for every $i = 1, \ldots, m'$.

($\Rightarrow$) Since the number of jobs $n = 2m$, by Property 6.1.1 there are exactly two jobs scheduled on each machine. For each machine $M_i \in M$, place the elements corresponding to the jobs scheduled on machine M_i along with associated element $z_i \in Z$ in triple A_i. Since the bag constraints are satisfied in the schedule and $B_1 = X$ and $B_2 = Y$, triple A_i contains exactly one element from each of X, Y, and Z. Let jobs $J_j \in B_1$ and $J_{j'} \in B_2$ be scheduled on machine M_i, then the load of M_i is

$$\frac{p_j}{s_i} + \frac{p_{j'}}{s_i} = \frac{1}{s_i}(p_j + p_{j'}) \leq \beta \Leftrightarrow p_j + p_{j'} \leq \beta s_i = \beta - s(z_i)$$

$$\Rightarrow p_j + p_{j'} + s(z_i) = s(a_j) + s(a_{j'}) + s(z_i) \leq \beta. \tag{6.1}$$

Assume that the load of at least one machine M_i is strictly less than β. Under this assumption, sum inequality (6.1) over all machines $M_1, M_2, \ldots, M_m$ to get

$$\sum_{a_j \in X \cup Y \cup Z} s(a_j) < m\beta = m'\beta,$$

which cannot happen. Therefore, for every triple A_i, $\sum_{a_j \in A_i} s(a_j) = \beta$.

($\Leftarrow$) Construct a schedule for the scheduling instance as follows. Let jobs j_i, j_i' correspond to elements $x_i \in X$ and $y_i \in Y$ in triple A_i. Schedule jobs j_i and j_i' on the machine M_i associated with element $z_i \in Z$ in A_i, $i = 1, \ldots, m$. The bags are $B_1 = X$ and $B_2 = Y$, so the bag constraints are clearly satisfied. Since

$$s(x_i) + s(y_i) + s(z_i) = \beta \Rightarrow p_{j_i} + p_{j_i'} = \beta - s(z_i) \Leftrightarrow \frac{1}{s_i}(p_{j_i} + p_{j_i'})$$

$$= \frac{\beta - s(z_i)}{s_i} = \beta,$$

hence, the makespan is β. ∎

As a note, Yu *et al.* [102] proved that the special case of the numerical 3-dimensional matching problem where two of the three disjoint sets $X = \{a_1, a_2, \ldots, a_{m'}\}$ and $Y = \{a_{m'+1}, a_{m'+2}, \ldots, a_{2m'}\}$ have elements with sizes $s(a_1) = s(a_{m'+1}) = 1$, $s(a_2) = s(a_{m'+2}) = 2, \ldots, s(a_{m'}) = s(a_{2m'}) = m'$ is strongly NP-hard. Our reduction still works in this case and because $B_1 = X$, $B_2 = Y$, and $m = m'$, one can show even with a limited set of job lengths $p_j \in \{1, 2, \ldots, m\}$, $Q|bag|C_{max}$ with $\mathscr{b} = 2$ bags is strongly NP-hard.

Corollary 6.6.7 *$Q|bag, p_j \in \{1, 2, \ldots, m\}|C_{max}$ with $\mathscr{b} = 2$ bags is strongly* NP*-hard.*

Chapter 7

Conclusions

Closing the 3/2-to-2 hardness gap of $R||C_{max}$ remains an open problem in approximation algorithms and scheduling theory despite almost thirty years of research. In this thesis we present results from our investigation of special cases of $R||C_{max}$ and related hard scheduling problems. We improved upon several previously best-known approximation algorithms and studied the structure of some of these problems in the hopes of revealing previously unknown complexities surrounding the 3/2-to-2 hardness gap of $R||C_{max}$. While the research presented in this thesis did not reconcile the outstanding 3/2-to-2 hardness gap, we discovered new approximation algorithms with tight approximation ratios (assuming $\mathsf{P} \neq \mathsf{NP}$) for some variants of $R||C_{max}$. In some cases these problems share the same 3/2-lower bound on the approximation ratio of $R||C_{max}$.

Below we provide a summary of our results in Section 7.1, and briefly discuss future work and open problems in Section 7.2.

7.1 Summary

In Chapter 3 we gave 3/2-approximation algorithms for the graph balancing problem with two job lengths, matching the 3/2-hardness lower bound known for the approximation ratio for this problem. The best-known approximation algorithm prior to our results for the graph balancing problem with two job lengths had approximation ratio 1.652 [66].

In Chapter 4 we proved there exists a $(\sqrt{65} + 7)/8$-approximation algorithm for the graph balancing problem with two speeds and two job lengths, this to the best of our knowledge is the first approximation algorithm with approximation ratio strictly less than 2 for a scheduling problem on uniform parallel machines that shares the same 3/2-inapproximability bound as $R||C_{max}$. We also present a $(2 - \ell_s/\ell_b)$-approximation algorithm for the restricted assignment problem with two job lengths $p_j \in \{\ell_s, \ell_b\}$ on uniform parallel machines, where $\ell_s < \ell_b$. Previous to this, the best-known approximation algorithms for both problems were those for $R||C_{max}$ and have ratios of 2.

In Chapter 5 we prove that unless $\mathsf{P} = \mathsf{NP}$ there is no ρ-approximation algorithm with $\rho < 3/2$ for the graph balancing problem restricted to instances where the job-intersection graph is diamondless, even if the job lengths are only 1 or 2. This was accomplished by strengthening a reduction by Ebenlendr *et al.* [21]. For $R||C_{max}$ with bounded job assignments,

we gave a flow-based approximation algorithm that has approximation ratio $5/3$ for $r = 4$, $3/2$ for $r = 3$, and 1 for $r = 2$. This approximation algorithm is also a $3/2$-approximation algorithm for $R||C_{max}$ restricted to diamondless job-intersection graphs, and solves $R||C_{max}$ restricted to triangle-free graphs. By combining our algorithm with the best-known algorithms for the restricted assignment problem and $R||C_{max}$, we designed a $(2-1/(r-1))$-approximation algorithm for the restricted assignment problem with two job lengths when $r \geq 3$.

In Chapter 6 we presented results for $R|bag|C_{max}$ and special cases of this problem with an emphasis placed on when the number of bags is small. First, we gave a simple b-approximation algorithm for $R|bag|C_{max}$ and a PTAS for $R|bag|C_{max}$ when there the numbers of machine types and bags are both constant; the latter result implies there is a PTAS for $Q|bag|C_{max}$ when both the number of machine speeds and number of bags are constant. Next, we gave a $b/2$-approximation algorithm for the graph balancing problem with $b \geq 2$ bags; this approximation ratio is tight for $b = 2$ bags and for $b = 3$ bags, unless $\mathsf{P} = \mathsf{NP}$. Following this, we presented polynomial-time algorithms for the restricted assignment problem with bags on uniform parallel machines where all the jobs have unit length ($Q|bag, \mathcal{M}_j, p_j = 1|C_{max}$) and for $P|bag|C_{max}$ with $b = 2$ bags. Then, we proved that unless $\mathsf{P} = \mathsf{NP}$ there are no ρ-approximation algorithms with $\rho < 3/2$ for the restricted assignment and graph balancing problems with $b = 2$ and $b = 3$ bags, respectively. Finally, we proved that $Q|bag|C_{max}$ with $b = 2$ bags is strongly NP-hard; this implies there is no FPTAS for $Q|bag|C_{max}$ when there are two bags, unless $\mathsf{P} = \mathsf{NP}$.

7.2 Future Work and Open Problems

We conclude this thesis by presenting some future work and open problems. For the sake of brevity, we only state open work that directly results from our research presented in this thesis. We give these problems by chapter.

Chapter 3. There still remains a $3/2$-to-$7/4$ hardness gap for the graph balancing problem. As we show in Chapter 3, there are $3/2$-approximation algorithms for the graph balancing problem with two job lengths. In fact, since our research two independent research groups have also developed $3/2$-approximation algorithms [11, 45]; both of these works use different techniques than the algorithm we presented in Chapter 3.1. Our algorithm described in Chapter 3.2 uses an approach that is somewhat similar to that in [11]. A natural special case of the the graph balancing problem to consider is the graph balancing problem with a constant number $c > 2$ job lengths and an open question is whether approximation algorithms for this problem with approximation ratio less than $7/4$ exist.

Chapter 4. We broke the 2-approximation barrier for the graph balancing problem with two speeds and two job lengths by giving an approximation algorithm with approximation ratio $(\sqrt{65}+7)/8 \approx 1.88278$. It is unclear if the approximation ratio we obtained can be improved upon. It does not appear the techniques we used can generalize easily to three or more machine speeds, and some of the techniques that we use rely on the assumption that there are only two job lengths. Further investigation may be necessary for the graph balancing problem with speeds for a fixed number of machine speeds and/or job lengths.

Chapter 5. We studied $R||C_{max}$ with simple job-intersection structure, however, the complexity of $R||C_{max}$ restricted to planar job-intersection graphs remains open. Our $3/2$-hardness proof for the diamondless case does not necessarily yield instances of $R||C_{max}$ with planar

job-intersection graphs. Planar graphs can admit diamonds, so we suspect it may not be polynomial-time solvable.

Chapter 6. We developed several results for both $R|bag|C_{max}$ and special cases of this problem. Currently there are several outstanding problems.

First, we proved that unless $\mathsf{P} = \mathsf{NP}$ there is no ρ-approximation algorithm with $\rho < 3/2$ for the graph balancing problem with $\mathscr{b} = 3$ bags unless $\mathsf{P} = \mathsf{NP}$, and our $\mathscr{b}/2$-approximation for this problem matches this lower bound for $\mathscr{b} = 3$ bags; beyond this, it is not clear how the hardness gap may differ for any number of bags. Assuming $\mathsf{P} = \mathsf{NP}$, there is no constant-factor approximation algorithm for the restricted assignment problem with bags, presently it is unknown whether or not the same is true for the graph balancing problem with bags.

We gave a PTAS for $R|bag|C_{max}$ with a constant numbers of machine types and bags. Das and Wiese [17] designed a PTAS for $P|bag|C_{max}$ and [31, 52] present PTASs for $R||C_{max}$ with a constant number of machine types. It is unclear if there is a PTAS for $R|bag|C_{max}$ when only the number of machine types is constant. It is worth noting that when the number of machine types is not constant, $R|bag|C_{max}$ with a constant number of bags has no PTAS (Corollary 6.6.2, Corollary 6.6.4). Currently we do not know whether there is a PTAS for makespan minimization on uniform parallel machines with bags ($Q|bag|C_{max}$) when the number of machine speeds is constant.

As there are no constant-factor approximation algorithms for the restricted assignment problem with bags assuming $\mathsf{P} \neq \mathsf{NP}$, an interesting angle for research is to investigate other special cases of this problem, like parallel machine scheduling with processing set restrictions and bags. As we discussed in Section 2.2.2, scheduling problems on parallel machines with processing set restrictions have approximation algorithms with approximation ratios less than 2; once the bags are introduced, there may exist constant-factor approximation algorithms. To the best of our knowledge, parallel machine scheduling with processing set restrictions and bags has not yet been investigated in the literature.

Bibliography

[1] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[2] Y. Asahiro, J. Jansson, E. Miyano, H. Ono, and K. Zenmyo. Approximation algorithms for the graph orientation minimizing the maximum weighted outdegree. *Journal of Combinatorial Optimization*, 22(1):78–96, 2011.

[3] Y. Asahiro, E. Miyano, and H. Ono. Graph classes and the complexity of the graph orientation minimizing the maximum weighted outdegree. *Discrete Applied Mathematics*, 159(7):498–508, 2011.

[4] Y. Asahiro, E. Miyano, H. Ono, and K. Zenmyo. Graph orientation algorithms to minimize the maximum outdegree. *International Journal of Foundations of Computer Science*, 18(02):197–215, 2007.

[5] N. Bansal and S. Khot. Optimal long code test with one free bit. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 453–462. IEEE, 2009.

[6] N. Bansal and M. Sviridenko. The santa claus problem. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of Computing*, pages 31–40. ACM, 2006.

[7] H. Bodlaender, K. Jansen, and G. Woeginger. Scheduling with incompatible jobs. *Discrete Applied Mathematics*, 55(3):219–232, 1994.

[8] A. Brandstädt, V.B. Le, and J. Spinrad. *Graph classes: a survey*. SIAM, 1999.

[9] G. Brodal and R. Fagerberg. Dynamic representations of sparse graphs. In *Workshop on Algorithms and Data Structures*, pages 342–351. Springer, 1999.

[10] D. Chakrabarty, S. Khanna, and S. Li. On (1, ε)-restricted assignment makespan minimization. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1087–1101. SIAM, 2015.

[11] D. Chakrabarty and K. Shiragur. Graph balancing with two edge types. *arXiv preprint arXiv:1604.06918*, 2016.

[12] P-Y Chang, P. Damodaran, and S. Melouk. Minimizing makespan on parallel batch processing machines. *International Journal of Production Research*, 42(19):4211–4220, 2004.

[13] C. Chekuri and S. Khanna. On multidimensional packing problems. *SIAM journal on computing*, 33(4):837–851, 2004.

[14] L. Chen, K. Jansen, W. Luo, and G. Zhang. An efficient PTAS for parallel machine scheduling with capacity constraints. In *International Conference on Combinatorial Optimization and Applications*, pages 608–623. Springer, 2016.

[15] L. Chen, D. Marx, D. Ye, and G. Zhang. Parameterized and Approximation Results for Scheduling with a Low Rank Processing Time Matrix. In *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*, volume 66 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:14, 2017.

[16] F. Chudak and D. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *Journal of Algorithms*, 30:323–343, 1999.

[17] S. Das and A. Wiese. On minimizing the makespan when some jobs cannot be assigned on the same machine. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 87. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[18] E. Davis and J. Jaffe. Algorithms for scheduling tasks on unrelated processors. *Journal of the ACM (JACM)*, 28(4):721–736, 1981.

[19] T. Dokka, A. Kouvela, and F. Spieksma. Approximating the multi-level bottleneck assignment problem. *Operations Research Letters*, 40:282–286, 2012.

[20] T. Ebenlendr, M. Krčál, and J. Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. *Algorithmica*, 68(1):62–80, 2014.

[21] T. Ebenlendr, M. Krčál, and J. Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. In *Proceedings of the Nineteenth Annual ACM–SIAM Symposium on Discrete Algorithms (SODA)*, pages 483–490, 2008.

[22] F. Eisenbrand, K. Kesavan, R. Mattikalli, M. Niemeier, A. Nordsieck, M. Skutella, J. Verschae, and A. Wiese. Solving an avionics real-time scheduling problem by advanced IP-methods. In *European Symposium on Algorithms*, pages 11–22. Springer, 2010.

[23] L. Epstein and A. Levin. Scheduling with processing set restrictions: PTAS results for several variants. *International Journal of Production Economics*, 133(2):586–595, 2011.

[24] G. Even, M. Halldórsson, L. Kaplan, and D. Ron. Scheduling with conflicts: online and offline algorithms. *Journal of Scheduling*, 12(2):199–224, 2009.

[25] L. Fanjul-Peyro and R. Ruiz. Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 207(1):55–69, 2010.

[26] L. Fanjul-Peyro and R. Ruiz. Size-reduction heuristics for the unrelated parallel machines scheduling problem. *Computers & Operations Research*, 38(1):301–309, 2011.

[27] M. Gairing, B. Monien, and A. Woclaw. A faster combinatorial approximation algorithm for scheduling unrelated parallel machines. *Theoretical Computer Science*, 380(1):87–99, 2007.

[28] M. Garey and D. Johnson. "Strong" NP-completeness results: Motivation, examples, and implications. *Journal of the ACM (JACM)*, 25(3):499–508, 1978.

[29] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman New York, 1979.

[30] W. Gasarch. Guest column: The second P=?NP poll. *ACM SIGACT News*, 43(2):53–77, 2012.

[31] J. Gehrke, K. Jansen, S. Kraft, and J. Schikowski. A PTAS for scheduling unrelated machines of few different types. *International Journal of Foundations of Computer Science*, 2018.

[32] M. Ghirardi and C. Potts. Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *European Journal of Operational Research*, 165(2):457–467, 2005.

[33] C. Glass and H. Kellerer. Parallel machine scheduling with job assignment restrictions. *Naval Research Logistics (NRL)*, 54(3):250–257, 2007.

[34] M. Goemans and T. Rothvoß. Polynomiality for bin packing with a constant number of item types. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 830–839. SIAM, 2014.

[35] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the ACM (JACM)*, 23(4):665–679, 1976.

[36] K. Grage, K. Jansen, and K. Klein. An eptas for machine scheduling with bag-constraints. *arXiv preprint arXiv:1810.07510*, 2018.

[37] R. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technological Journal*, 45(9):1563–1581, 1966.

[38] R. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[39] R. Graham, E. Lawler, J. Lenstra, and K. Rinnooy. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[40] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2. Springer Science & Business Media, 2012.

[41] P. Hall. On representatives of subsets. *J. London Math. Soc*, 10(1):26–30, 1935.

[42] D. Hochbaum and D. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1):144–162, 1987.

[43] D. Hochbaum and D. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.

[44] J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.

[45] C. Huang and S. Ott. A combinatorial approximation algorithm for graph balancing with light hyper edges. In *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49:1–49:15, 2016.

[46] Y. Huo and J. Leung. Fast approximation algorithms for job scheduling with processing set restrictions. *Theoretical Computer Science*, 411(44-46):3947–3955, 2010.

[47] Y. Huo and J. Leung. Parallel machine scheduling with nested processing set restrictions. *European Journal of Operational Research*, 204(2):229–236, 2010.

[48] O. Ibarra and C. Kim. Heuristic algorithms for scheduling independent tasks on non-identical processors. *Journal of the ACM (JACM)*, 24(2):280–289, 1977.

[49] K. Jansen. An EPTAS for scheduling jobs on uniform processors: using an milp relaxation with a constant number of integral variables. *SIAM Journal on Discrete Mathematics*, 24(2):457–485, 2010.

[50] K. Jansen, K. Klein, and J. Verschae. Closing the gap for makespan scheduling via sparsification techniques. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[51] K. Jansen, K. Land, and M. Maack. Estimating the makespan of the two-valued restricted assignment problem. *Algorithmica*, 80(4):1357–1382, 2018.

[52] K. Jansen and M. Maack. An EPTAS for scheduling on unrelated machines of few different types. In *Workshop on Algorithms and Data Structures*, pages 497–508. Springer, 2017.

[53] K. Jansen, M. Maack, and R. Solis-Oba. Structural parameters for scheduling with assignment restrictions. In *International Conference on Algorithms and Complexity*, pages 357–368. Springer, 2017.

[54] K. Jansen, M. Maack, and R. Solis-Oba. Structural parameters for scheduling with assignment restrictions. *CoRR*, abs/1701.07242, 2017.

[55] K. Jansen and L. Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. *Mathematics of Operations Research*, 26(2):324–338, 2001.

[56] K. Jansen and L. Rohwedder. On the configuration-LP of the restricted assignment problem. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2670–2678. SIAM, 2017.

[57] K. Jansen and L. Rohwedder. A quasi-polynomial approximation for the restricted assignment problem. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 305–316. Springer, 2017.

[58] K. Jansen and L. Rohwedder. Compact LP relaxations for allocation problems. In *1st Symposium on Simplicity in Algorithms (SOSA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[59] K. Jansen and L. Rohwedder. Local search breaks 1.75 for graph balancing. *arXiv preprint arXiv:1811.00955*, 2018.

[60] K. Jansen and L. Rohwedder. On integer programming and convolution. In *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[61] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[62] R. Karp. *Reducibility Among Combinatorial Problems*. Springer, 1972.

[63] L. Khachiyan. A polynomial time algorithm in linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979.

[64] T. Kloks, D. Kratsch, and H. Müller. Finding and counting small induced subgraphs efficiently. *Information Processing Letters*, 74(3-4):115–121, 2000.

[65] D. Knop and M. Koutecký. Scheduling meets n-fold integer programming. *Journal of Scheduling*, 21(5):493–503, 2018.

[66] S. Kolliopoulos and Y. Moysoglou. The 2-valued case of makespan minimization with assignment constraints. *Information Processing Letters*, 113(1):39–43, 2013.

[67] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 5th edition, 2012.

[68] V. Kumar, M. Marathe, S. Parthasarathy, and A. Srinivasan. Scheduling on unrelated machines under tree-like precedence constraints. *Algorithmica*, 55:205–226, 2005.

[69] E. Lawler and J. Labetoulle. On preemptive scheduling of unrelated parallel processors by linear programming. *Journal of the ACM (JACM)*, 25(4):612–619, 1978.

[70] K. Lee, J. Leung, and M. Pinedo. A note on graph balancing problems with restrictions. *Information Processing Letters*, 110(1):24–29, 2009.

[71] K. Lee, J. Leung, and M. Pinedo. Makespan minimization in online scheduling with machine eligibility. *Annals of Operations Research*, 204(1):189–222, 2013.

[72] J. Lenstra, D. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46(1–3):259–271, 1990.

[73] J. Leung and C. Li. Scheduling with processing set restrictions: A survey. *International Journal of Production Economics*, 116(2):251–262, 2008.

[74] J. Leung and C. Li. Scheduling with processing set restrictions: A literature update. *International Journal of Production Economics*, 175:1–11, 2016.

[75] J. Leung and C. T. Ng. Fast approximation algorithms for uniform machine scheduling with processing set restrictions. *European Journal of Operational Research*, 260(2):507–513, 2017.

[76] C. Li and X. Wang. Scheduling parallel machines with inclusive processing set restrictions and job release times. *European Journal of Operational Research*, 200(3):702–710, 2010.

[77] S. Li. Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 283–294. IEEE, 2017.

[78] S. Li, G. Li, and S. Zhang. Minimizing makespan with release times on identical parallel batching machines. *Discrete Applied Mathematics*, 148(1):127–134, 2005.

[79] Y. Lin and W. Li. Parallel machine scheduling of machine-dependent jobs with unit-length. *European Journal of Operational Research*, 156(1):261–266, 2004.

[80] E. Mokotoff and P. Chrétienne. A cutting plane algorithm for the unrelated parallel machine scheduling problem. *European Journal of Operational Research*, 141(3):515–525, 2002.

[81] G. Muratore, U. Schwarz, and G. Woeginger. Parallel machine scheduling with nested job assignment restrictions. *Operations Research Letters*, 38(1):47–50, 2010.

[82] J. Orlin. Max flows in $O(nm)$ time, or better. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, pages 765–774. ACM, 2013.

[83] J. Ou, J. Leung, and C. Li. Scheduling parallel machines with inclusive processing set restrictions. *Naval Research Logistics (NRL)*, 55(4):328–338, 2008.

[84] D.R. Page and R. Solis-Oba. A 3/2-approximation algorithm for the graph balancing problem with two weights. *Algorithms*, 9(2):38, 2016.

[85] D.R. Page and R. Solis-Oba. Approximation algorithms for the graph balancing problem with two speeds and two job lengths. *Journal of Combinatorial Optimization*, https://doi.org/10.1007/s10878-018-0339-x, 2018.

[86] D.R. Page and R. Solis-Oba. Makespan minimization on unrelated parallel machines with a few bags. In *International Conference on Algorithmic Aspects in Information and Management*, pages 24–35. Springer, 2018.

[87] D.R. Page, R. Solis-Oba, and M. Maack. Makespan minimization on unrelated parallel machines with simple job-intersection structure and bounded job assignments. In *International Conference on Combinatorial Optimization and Applications*, pages 341–356. Springer, 2018.

[88] S. Plotkin, D. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301, 1995.

[89] A. Punnen and K.P.K. Nair. Improved complexity bound for the maximum cardinality bottleneck bipartite matching problem. *Discrete Applied Mathematics*, 55(1):91–93, 1994.

[90] B. Saha and A. Srinivasan. A new approximation technique for resource-allocation problems. In *Proceedings of the 1st Annual Symposium on Innovations in Computer Science*, pages 342–357, 2010.

[91] P. Schuurman and G. Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling*, 2(5):203–213, 1999.

[92] E. Shchepin and N. Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, 33:127–133, 2005.

[93] D. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(1-3):461–474, 1993.

[94] D. Sule. *Production planning and industrial scheduling: examples, case studies and applications*. CRC press, 2007.

[95] O. Svensson. Conditional hardness of precedence constrained scheduling on identical machines. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, pages 745–754. ACM, 2010.

[96] O. Svensson. Santa Claus schedules jobs on unrelated machines. In *Proceedings of the Forty-Third ACM Symposium on Theory of Computing*, pages 617–626, New York, 2011. ACM.

[97] J. Verschae and A. Wiese. On the configuration-LP for scheduling on unrelated machines. *Journal of Scheduling*, 17(4):371–383, 2014.

[98] V. Vizing. On an estimate of the chromatic class of a p-graph. *Diskret analiz*, 3:25–30, 1964.

[99] C. Wang and R. Sitters. On some special cases of the restricted assignment problem. *Information Processing Letters*, 116(11):723–728, 2016.

[100] D. Williamson and D. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[101] L. Yu, H. Shih, M. Pfund, W.M. Carlyle, and J. Fowler. Scheduling of unrelated parallel machines: an application to pwb manufacturing. *IIE transactions*, 34(11):921–931, 2002.

[102] W. Yu, H. Hoogeveen, and J. Lenstra. Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard. *Journal of Scheduling*, 7(5):333–348, 2004.

[103] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, pages 681–690. ACM, 2006.

Curriculum Vitae

Name:	Daniel Page
Post-Secondary Education and Degrees:	University of Manitoba Winnipeg, MB 2007–2011 B.C.Sc. (Honours) University of Manitoba Winnipeg, MB 2012–2014 M.Sc. University of Western Ontario London, ON 2015–2019 Ph.D.
Related Work Experience:	Limited-Duties Instructor The University of Western Ontario 2019 Teaching Assistant The University of Western Ontario 2015–2018 Sessional Instructor The University of Manitoba 2012–2014 Museum Interpreter The Royal Aviation Museum of Western Canada Winnipeg, MB 2011–2014 Science Instructor Mad Science of Manitoba Winnipeg, MB 2011–2012

Inclusion Specialist
Sunny Mountain Daycare Centre
Winnipeg, MB
2007–2009

Publications:

Theses

Page, D.R. (2014). Tractability and approximability for subclasses of the makespan problem on unrelated parallel machines.

Page, D.R. (2011). Generalized methods for restricted weak composition enumeration.

Journal Publications

Page, D.R., Solis-Oba, R. (2018). Approximation algorithms for the graph balancing problem with two speeds and two job lengths. *Journal of Combinatorial Optimization*, https://doi.org/10.1007/s10878-018-0339-x.

Page, D.R., Solis-Oba, R. (2016). A 3/2-approximation algorithm for the graph balancing problem with two weights. *Algorithms*, 9(2), 38, https://doi.org/10.3390/a9020038.

Page, D.R. (2015). Approximation algorithms for subclasses of the makespan problem on unrelated parallel machines with restricted processing times. *SOP Transactions on Applied Mathematics*, 2(1), 20–27, DOI: 10.15764/APHY.2015.01003.

Page, D.R. (2013). Parallel algorithm for second-order restricted weak integer composition generation for shared memory machines. *Parallel Processing Letters*, 23(3), 1350010, https://doi.org/10.1142/S0129626413500102.

Page, D.R. (2013). Generalized algorithm for restricted weak composition generation. *Journal of Mathematical Modelling and Algorithms in Operations Research*, 12(4), 345–372.

Refereed Conference Publications

Page, D.R., Solis-Oba, R. (2018). Makespan minimization on unrelated parallel machines with a few bags. *In: Algorithmic Aspects in Information and Management. AAIM 2018. Lecture Notes in Computer Science*, 11343, 24–35, https://doi.org/10.1007/978-3-030-04618-7_3.

Page, D.R., Solis-Oba, R., Maack, M. (2018). Makespan minimization on unrelated parallel machines with simple job-intersection structure and bounded job assignments. In: *Combinatorial Optimization and Applications. COCOA 2018. Lecture Notes in Computer Science*, 11346, 341–356, https://doi.org/10.1007/978-3-030-04651-4_23.

Honours and Awards:

Western University Experiential Opportunities Fund
2018

Ontario Graduate Scholarship
2017–2018

University of Western Ontario, Society of Graduate Students (SOGS)
SOGS Graduate Student Teaching Awards Nominee
2016–2018

University of Western Ontario Research in Computer Science (UWORCS) 2018
First Prize, *Data Mining, Machine Learning, AI & Theory of Computer Science*
2018

University of Western Ontario Research in Computer Science (UWORCS) 2017
First Prize, *Computer Algebra & Theory of Computer Science*
2017

University of Western Ontario Research in Computer Science (UWORCS) 2016
First Prize, *Theory of Computer Science, Computer Algebra & Symbolic Computation*
2016

University of Manitoba, University 1
University 1 Excellence in Teaching Award Nominee (2012/2013)
2013

University of Manitoba, Department of Computer Science
Special Department of Computer Science Award of Excellence
2013