Technische Universität München

Fakultät für Mathematik

Lehrstuhl für Operations Research

Min-Sum Set Cover, OR-Scheduling, and Related Problems

Felix Happach

Vollständiger Abdruck der von der Fakultät für Mathematik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr. Gero Friesecke
Prüfer der Dissertation:	1. Prof. Dr. Andreas S. Schulz
	2. Prof. Dr. Thomas Lidbetter (Rutgers University)

Die Dissertation wurde am 22.06.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Mathematik am 15.09.2020 angenommen.

Acknowledgments

During the past years as a doctoral candidate, I have had the pleasure of meeting with and benefiting from many people both on a professional and on a personal level. I am grateful to all my friends, family, colleagues and other people who have accompanied me during these almost four years.

A special thanks goes, of course, to my supervisor Andreas S. Schulz who guided and supported me all these years. Whenever I knocked on his door, he provided me with helpful advice and new interesting problems, ideas and "low hanging fruit". He gave me the freedom to work on whichever topic I liked and, even though I started out working on a completely different problem in the beginning, I was attracted to the field of scheduling problems and approximation algorithms by his enthusiasm. His profound knowledge in the area and his encyclopedic recall of the literature and techniques aided me in various glassboard discussions and whenever I got stuck or needed some input.

I am also much obliged to my mentor Steffen Borgwardt who supported me continuously since and beyond my master thesis project. He has always had an open ear for questions of any kind. Especially, I very much appreciate him giving me the opportunity to visit the University of Colorado Denver for six weeks. This research stay was an experience from which I have benefited incredibly and that I will never forget (not only because it resulted in a joint paper, which, unfortunately, did not fit the topic of this thesis, but also personally).

Also, I want to thank Thomas Lidbetter, whom I had the pleasure to meet at MAPSP 2019, for the many fruitful discussions there and during his visit at TU Munich. His valuable input triggered new ideas and results on some open problems I had been struggling with for quite some time.

Moreover, I am thankful to all the current and former members of the Operations Research Group at TU Munich for being such great colleagues and friends. Thanks also to all the other members of the Discrete Mathematics, Optimization and Convexity Group (or, as I am still used to it, the "M9 Group") in Garching. In particular, I want to thank Matthias Brugger, Ulf Friedrich, Marinus Gottschau, Marcus Kaiser, Marilena Leichter, Clara Waldmann and Stefan Weltge for the fruitful discussions on common projects, helpful advice and joint brainstorming on "my problems" as well as the fun together beyond research and teaching. Further, I would like to thank my (former) office mates Max Fiedler, Diogo Poças and Alexandros Tsigonias-Dimitriadis.

Last but not least, I am deeply grateful to my wife Carmen for all her love and continuous support. Without you, I would not have been able to accomplish everything that I did, especially during the past few months. Thank you for always having my back! I love you!

Felix Happach

Abstract

We address various scheduling problems with OR-precedence constraints that are extensions of min-sum set cover, minimum latency set cover and generalized min-sum set cover. Using machinery from the theory of scheduling, we devise new exact and approximative algorithms for variants of these problems. We consider two main objective functions: makespan and total weighted completion time. For the makespan, we present an approximation algorithm, provide a lower bound on the approximation factor and present a polynomial time algorithm for the preemptive case. We study the relation between min-sum covering problems and OR-scheduling to minimize the total weighted completion time, and propose various algorithms, e.g., for laminar generalized min-sum set cover and a generalization of precedence-constrained min-sum set cover. Moreover, we present a framework for obtaining 4η-approximation algorithms for various linear ordering problems that generalize precedence-constrained single-machine scheduling. Some algorithms in this thesis are based on linear programs. We review classical LP formulations from the literature in the OR-scheduling context, characterize valid inequalities and analyze the integrality gaps of these relaxations. Finally, we discuss a different scheduling model, which is called concurrent open shop, and present a new approximation algorithm for the preemptive variant with release dates.

Zusammenfassung

Wir betrachten diverse Scheduling-Probleme mit OR-Vorgängerbeziehungen, welche Erweiterungen von *Min-Sum Set Cover*, *Minimum Latency Set Cover* und *Generalized Min-Sum Set Cover* sind. Mit Hilfe von Methoden aus der Schedulingtheorie leiten wir neue exakte und approximative Algorithmen für Varianten dieser Probleme her. Wir betrachten zwei der wichtigsten Zielfunktionen aus dem Gebiet des Scheduling: die gesamte Produktionsdauer und die Summe der gewichteten Fertigstellungszeiten. Für die Produktionsdauer liefern wir eine untere Schranke für die Approximierbarkeit und geben einen Approximationsalgorithmus für den allgemeinen Fall sowie einen polynomiellen Algorithmus für den Fall mit Job-Unterbrechungen an. Außerdem untersuchen wir den Zusammenhang zwischen *Min-Sum Set Cover* und OR-Scheduling und formulieren diverse Algorithmen u.a. für *Laminar Generalized Min-Sum Set Cover* sowie eine Verallgemeinerung von *Precedence-Constrained Min-Sum Set Cover*. Wir präsentieren ein Grundgerüst, um 4η-Approximationsalgorithmen für Verallgemeinerungen von Schedulingproblemen zu erhalten. Manche Algorithmen in dieser Arbeit basieren auf linearen Programmen. Wir analysieren klassische LP-Formulierungen aus der Literatur im OR-Scheduling Kontext, charakterisieren zulässige Ungleichungen und studieren die Ganzzahligkeitslücke dieser Relaxierungen. Schlussendlich betrachten wir ein weiteres Scheduling-Modell, *Concurrent Open Shop*, und geben einen neuen Approximationsalgorithmus für die Variante mit Unterbrechungen und Freigabezeitpunkten der Jobs an.

Contents

Chapter 1

Introduction

This thesis addresses various min-sum covering problems and related scheduling problems, studies their connection and presents algorithms therefor. All problems discussed in this thesis fall into the large class of *combinatorial optimization* problems, which are characterized as those problems where one "searches for an optimum object in a finite collection of objects" ([147], page 1). For a detailed study on combinatorial optimization, we refer to the textbooks of Korte and Vygen [102], Papadimitriou and Steiglitz [132] and Schrijver [147].

The basis of the models studied here is *min-sum set cover*, which was introduced by Feige, Lovász and Tetali [44, 45]. Min-sum set cover is a variant of the well-known *set cover* problem, which is one of the classical problems in combinatorial optimization [98]. In set cover, one intends to find a smallest collection of given sets over a finite ground set of elements that covers all elements. Min-sum set cover and the extensions considered in this thesis can be seen as variants of set cover, where one means to find a linear ordering of the elements that satisfies some constraints and is optimal with respect to some objective function. These problems are tackled by interpreting them as instances of *scheduling* problems with so-called *OR-precedence constraints*. Scheduling is an abstract framework for many real-world scenarios that go beyond simple assignment problems. In scheduling, one tries to find an optimal assignment of jobs to machines while (at the same time) attempting to obtain an optimal ordering of jobs allocated to the individual machines subject to certain constraints.

A prime example for a scheduling problem arises in, e.g., doctors' offices or in surgery every day. There are several patients arriving over time, but there is only a limited number of doctors and rooms available. The patients have varied needs that require a varied timespan, varied technical equipment, a varied number of persons present during surgery, etc. More generally, we have jobs (patients) with varied processing times that have to be assigned to a limited number of machines (rooms, doctors). The jobs might also compete for several resources (equipment, specific persons such as an anesthesiologist), have weights (some surgery might be more urgent than others), or are only available in a certain time interval, etc. The goal is to find an assignment of the jobs to the machines and an ordering of the jobs on each machine that minimizes the *sum of weighted completion times* ("urgent jobs preferably first").

Also the well-known *traveling salesperson problem* (TSP), one of the classical problems in combinatorial optimization [35], can be viewed as a scheduling problem. Here, we are given a set of n cities with pairwise symmetric non-negative distances, i.e., the cost of traveling from city i to city j equals the cost of traveling from j to i. The task is to find a tour of minimum cost that visits each city exactly once. (For more information on TSP as well as applications and solution approaches, we refer the reader to the textbook of Lawler et al. [110].) On a more abstract level, one can reformulate TSP as follows: Find a linear ordering of the cities of minimum cost, where the incurred cost of a city depends on the previously visited city. This is a scheduling problem with one machine where each city corresponds to a job, and the processing time of each job depends on the preceding job(s). The objective is to minimize the *makespan*, i.e., the completion time of the last job.

These examples only cover a very small class of scheduling models. In this thesis, we do not consider specific applications of scheduling problems, but rather conduct fundamental research. That is, we analyze these problems from an abstract way, and present basic algorithms to obtain optimal or approximately optimal feasible solutions. Before we introduce the models and the main concepts in Section 1.1, we want to give a brief overview of the historical development of (parts of) the field. For a more detailed historical overview of the area, we refer to [138].

Although the first scheduling problem was mentioned by Gantt [51] in 1913, one could say that the research area of scheduling was actually founded in the 1950's by the seminal papers of Johnson [95], Jackson [91], Smith [157] and McNaughton [123], see [138]. During the next decades, the number of different settings and results increased rapidly: Single-machine, parallel identical machines, unrelated machines, flow shop, job shop, (different types of) precedence-constraints, release dates, deadlines, several objective functions, etc.[1] In 1979, some of the major researchers in the field published an influential survey [68] in which they introduced a unified notation to classify the different scheduling models.

In the 1960's and 1970's, research focused on obtaining efficient algorithms for certain scheduling problems or showing that such algorithms cannot exist under some complexity assumptions, see Section 1.1.1. The goal of many papers in the field, such as, e.g., [86, 47, 129, 130, 32, 85, 162, 23, 113, 52, 108, 112], was to distinguish the "border" between "efficiently solvable problems" and "hard problems". For those problems that were classified to be "hard", research then concentrated on finding guarantees for provably good approximate solutions, see, e.g., [66, 67, 144, 107, 84, 115, 148, 71]. One paper that should be mentioned explicitly is the seminal paper of Graham [66]. Graham introduced a very simple algorithm called *list scheduling* that, to this day, is used as a framework for many scheduling algorithms. He further proved that this algorithm achieves provably good solutions for a very basic scheduling problem, thereby presenting one of the first *approximation algorithms* in scheduling [138].

[1] See Section 1.1.4 for an explanation of some of these terms.

Note that the above references are not exhaustive and only comprise very restricted classes of *deterministic scheduling models* that are most closely related to the scheduling models studied in this thesis. We introduce these models in Sections 1.1.4 and 1.2.2. Another popular research direction is *stochastic scheduling*, where some of the input data is given by probability distributions, but which is not considered here. For an extensive survey on scheduling problems, we refer to, e.g., [111, 27, 136, 114].

In this thesis, we focus on various deterministic scheduling problems, mostly where jobs are subject to a specific type of *precedence constraints*, see Chapters 2 to 4, and where jobs consist of different tasks, see Chapter 5. We review the relevant literature and introduce the problem specific tools and preliminaries at the beginning of the respective chapters.

1.1 Notation and Preliminaries

This section lays the foundation for the notation and terminology used throughout the thesis. The set of real numbers, integers and positive integers is denoted by $\mathbb{R}$, $\mathbb{Z}$ and $\mathbb{N}$, respectively. Further, we let $\mathbb{N}_0 \coloneqq \mathbb{N} \cup \{0\}$ be the set of non-negative integers, and define $[n] \coloneqq \{1, \dots, n\}$ for any $n \in \mathbb{N}$ and $[0] \coloneqq \emptyset$. If a set N_1 is a subset of a set N_2 and they might coincide, we denote this by $N_1 \subseteq N_2$. If N_1 is a strict subset of N_2, we write $N_1 \subsetneq N_2$. The cardinality of a set N is denoted by $|N|$ and its power set is $2^N \coloneqq \{S \mid S \subseteq N\}$.

1.1.1 Complexity and Approximation Algorithms

A key research area in combinatorial optimization is the design and analysis of *algorithms* and the closely related *complexity theory*. When confronted with an *optimization problem* Π, we intend to design an algorithm that returns an optimal solution for any instance of Π. We mainly focus on minimization problems, where an optimal solution is a feasible solution of lowest objective function value. For maximization problems, we seek a feasible solution with highest possible objective value. Theoretically, finding an optimal solution is often possible by, e.g., enumerating all feasible solutions and picking the best one. In practice, however, the number of feasible solutions might be huge compared to the actual instance of the problem. We simplify all notions and concepts in this section as far as possible to improve readability. For details, we refer to the textbook of Garey and Johnson [53].

For every optimization problem ("What is an optimal solution for instance $\mathcal{I}$?"), there is a corresponding *decision variant*. As input, the decision problem receives an integer $K \in \mathbb{Z}$, in addition to the instance $\mathcal{I}$, and asks whether there is a feasible solution of $\mathcal{I}$ with objective value at most K (for minimization) or at least K (for maximization), respectively. Depending on whether the answer to the decision problem is "yes" or "no", we call the instance a *Yes-instance* or *No-instance*, respectively.

Example 1.1 (Decision Variant for TSP)
The optimization version of TSP is "What is the minimal cost of a tour that visits each city exactly once?", and the corresponding decision version is "Is there a tour that visits each city exactly once of cost at most K?".

It is clear that, if we can solve an optimization problem, we can also solve the corresponding decision version: If we have an optimal solution for an instance, we can easily compare its objective value to K. On the other hand, if the decision variant is "hard", then the optimization problem is also "hard". Therefore, instead of considering the optimization problem itself, we can often restrict to the corresponding decision variant.

Running Time. The main quantitative measures for analyzing algorithms are the input size of an instance and the running time of an algorithm. The *input size* of an instance $\mathcal{I}$ of an optimization problem Π is the number of bits needed to store the information that defines $\mathcal{I}$. The *running time* of an algorithm is the number of elementary operations that the algorithm needs until it returns a feasible solution. Typically, the running time is measured by a function $g(n)$ that indicates the number of elementary operations the algorithm requires in the *worst-case* until in terminates on an instance of size at most n.

In theoretical computer science, one often does not consider the exact running time, but focuses on upper and lower bounds. A helpful tool for this is the $\mathcal{O}$-notation. Let $g_1, g_2 : \mathbb{N} \to \mathbb{N}$ be two functions. We write $g_1 \in \mathcal{O}(g_2)$ if there are $n_0, c \in \mathbb{N}$ such that $g_1(n) \leq c\, g_2(n)$ for all $n \geq n_0$. Conversely, we write $g_2 \in \Omega(g_1)$ if $g_1 \in \mathcal{O}(g_2)$. A standard notion in theoretical computer science is *polynomial running time*.

Definition 1.2 (Polynomial-Time Algorithm)
Let Π be an optimization problem. An algorithm Alg has *polynomial running time* if there is a polynomial $g : \mathbb{N} \to \mathbb{N}$ such that, for every instance $\mathcal{I}$ of Π of size at most n, the worst-case running time of Alg on $\mathcal{I}$ is in $\mathcal{O}(g(n))$. If Alg returns an optimal solution for every instance $\mathcal{I}$ of Π and has polynomial running time, we call it a *polynomial-time algorithm* for Π.

Complexity Classes. We say an decision problem Π is *solvable in polynomial time* if there is a polynomial-time algorithm for Π. The set of polynomial-time solvable optimization problems is denoted by the complexity class P. Unfortunately, most problems in combinatorial optimization probably do not lie in this class, but in a larger class called NP. The complexity class NP is defined as the set of problems for which a *non-deterministic* polynomial-time algorithm exists. Roughly speaking, a problem Π is in NP if we can check in polynomial time whether a given solution is feasible for the decision version of Π. Note that $\mathrm{P} \subseteq \mathrm{NP}$.

We say that a problem Π' *reduces* to Π if there is a polynomial-time algorithm that maps instances of Π' to instances of Π, and has the property that the input instance is a YES-instance of Π' if and only if the output instance is a YES-instance of Π. That is, we can express Π' as a special case of Π. If Π' reduces to Π then Π is "harder" than Π' in some sense. A problem Π is called *NP-complete* if it is contained in NP and every $\Pi' \in$ NP reduces to Π. We say a problem Π is *NP-hard* if there is an NP-complete problem Π' that reduces to Π. Note that if Π' reduces to Π and Π reduces to $\overline{\Pi}$, then Π' reduces to $\overline{\Pi}$, i.e., reductions are transitive. A problem is *strongly NP-hard* if it remains NP-hard, even if all numerical parameters are bounded by a polynomial in the input dimension [52].

In 1971, Cook [33] identified the first NP-complete problem: SATISFIABILITY, which asks whether there is a true/false assignment of variables such that a given boolean formula is satisfied. In a seminal paper, Karp [98] showed that many standard combinatorial optimization problems, such as, e.g., CLIQUE, SET COVER, EXACT COVER and PARTITION, are NP-complete.[2] He presented a list of 21 problems that, to this day, are used to show NP-hardness of optimization problems. The results of Cook [33] and Karp [98] led to a number of follow-up works that classified many combinatorial optimization problems to be either in P, i.e., "easy problems", or NP-hard, i.e., "hard problems". It is a long outstanding open question in theoretical computer science and mathematics whether the complexity classes P and NP coincide, i.e., whether there is an NP-complete problem that is contained in P.[3] The common conjecture is that P $\neq$ NP.

Approximation Algorithms. Unless P = NP, we cannot hope to find polynomial-time algorithms for NP-hard optimization problems. One way to circumvent this is to relax the condition of optimality and focus on approximately optimal solutions instead.

Definition 1.3 (Approximation Algorithm)
Let Π be a minimization problem, ALG a polynomial-time algorithm and $\eta \geq 1$. For every instance $\mathcal{I}$ of Π, let $\textsc{Opt}(\mathcal{I})$ and $\textsc{Alg}(\mathcal{I})$ denote the optimal objective value of $\mathcal{I}$ and the objective value of the solution returned by ALG, respectively. We call ALG an *η-approximation algorithm* for Π if

$$\frac{\textsc{Alg}(\mathcal{I})}{\textsc{Opt}(\mathcal{I})} \leq \eta \quad \text{for all instances } \mathcal{I} \text{ of } \Pi. \tag{1.1}$$

If Π is a maximization problem, we call ALG an *η-approximation algorithm* if

$$\frac{\textsc{Opt}(\mathcal{I})}{\textsc{Alg}(\mathcal{I})} \leq \eta \quad \text{for all instances } \mathcal{I} \text{ of } \Pi. \tag{1.2}$$

[2] We refer to the appendix of [53] for an extensive list of NP-complete problems. The "classical" NP-hard problems of [98, 53] are written in capital letters throughout the thesis.

[3] www.claymath.org/millennium-problems/p-vs-np-problem (last checked: June 10, 2020)

Technically, one has to be careful about the signs of the objective values. For most combinatorial optimization problems, we can assume, w.l.o.g., that the objective value of any feasible solution is positive, i.e., the ratios in Definition 1.3 are well-defined. We call an approximation algorithm a *constant-factor approximation* if the factor η does not depend on the input size of the instance.

For $\eta \geq 1$ and $K \in \mathbb{N}$, the *approximate decision variant* of a minimization problem Π asks whether there is a feasible solution of objective value $\leq \eta K$ or if all feasible solutions have objective value strictly greater than K. Note that the approximate decision problem might return any answer if there is a feasible solution in the half-open interval $(K; \eta K]$. The approximate decision variant of a maximization problem is defined similarly and asks whether there is a feasible solution of objective value at least $\frac{1}{\eta} K$ or whether all solutions have objective value $< K$. Observe that, for $\eta = 1$, the approximate decision problem and the decision problem coincide. We say that an optimization problem Π is *NP-hard to η-approximate* if the corresponding approximate decision problem for η is NP-hard. An optimization problem Π is said to be *APX-hard* if there is a constant $\eta > 1$ such that it is NP-hard to η-approximate Π.

A $(1+\varepsilon)$-approximation algorithm whose running time is polynomial in the input size and $1/\varepsilon$ for any *fixed* $\varepsilon > 0$ is called a *polynomial-time approximation scheme* or *PTAS*. If the running time is polynomial in the size of the instance *and* in $1/\varepsilon$, it is called a *fully polynomial-time approximation scheme* or *FPTAS*. Note that there is no PTAS for APX-hard problems, unless P = NP. Garey and Johnson [52] showed that the existence of an FPTAS for a strongly NP-hard problem would imply P = NP. We refer to the textbooks of Hochbaum [83] and Williamson and Shmoys [165] and references therein for more details and examples on the design and analysis of approximation algorithms, as well as for concepts of proving hardness of approximation.

1.1.2 Graphs and Graph Classes

Graphs are useful combinatorial objects to encode dependencies and connections between elements of a finite set. The field of graph theory is a complete research area on its own from which we only need very few notions. We refer to [36] for an overview on basic graph theory, concepts and terminology, and to [124] for related basic algorithms and data structures.

A *directed graph*, or *digraph*, $G = (N, E)$ consists of a *set of nodes* N and a *set of arcs* $E \subseteq N \times N$. The nodes in $\{i \in N \mid (i, j) \in E\}$ are called *predecessors* of a node $j \in N$, and the *successors* of j are the nodes in $\{i \in N \mid (j, i) \in E\}$. A *subgraph* of G is a digraph $G' = (N', E')$ with $N' \subseteq N$ and $E' \subseteq E \cap (N' \times N')$. For a given set of nodes $S \subseteq N$, we call $G[S] \coloneqq (S, E \cap (S \times S))$ the *subgraph induced by* S. A *path* in G is a subset of nodes $\{i_1, \ldots, i_\ell\} \subseteq N$ such that $(i_q, i_{q+1}) \in E$ for all $q \in [\ell - 1]$, and if $i_1 = i_\ell$, we call $\{i_1, \ldots, i_\ell\}$ a *cycle*. We say a node i is *reachable* from a node j if there exists a path in G that starts at j and ends at i.

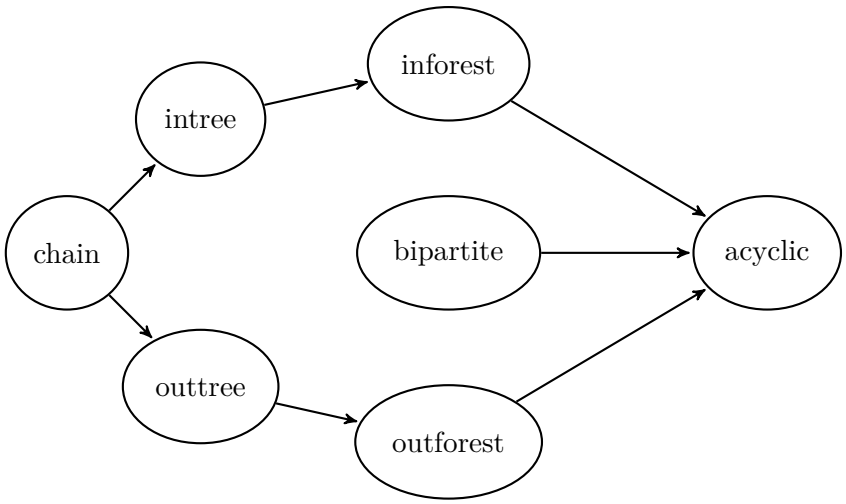

Fig. 1.1: Overview of the main digraph classes considered in this thesis. An arrow from class Λ_1 to Λ_2 indicates that Λ_1 is a special case of Λ_2. The directed graph depicted here is itself acyclic. If we remove the node "chain" or "acyclic" from the graph, we obtain an intree or outforest, respectively.

A *graph traversing algorithm* is an algorithm that starts at a given node in G and recursively enumerates all its successors, see, e.g., Chapter 9 of [124]. That is, a graph traversing algorithm enumerates all nodes that are reachable from the start node. Standard graph traversing algorithms are, e.g., breadth-first search or depth-first search that run in polynomial time in the size of the graph.

Graph Classes. A digraph $G = (N, E)$ is called *acyclic* if it does not contain a cycle, and *bipartite* if the set of nodes can be partitioned into $N = A \dot\cup B$ such that $E \subseteq A \times B$. If every node of a directed acyclic graph has at most one successor, we call the digraph an *inforest*. Similarly, an *outforest* is a directed acyclic graph where every node has at most one predecessor. A node of an inforest or outforest is called a *root* if it has no successors or predecessors, respectively. An inforest with only one root is called an *intree*. Similarly, an outforest that contains only one root is called an *outtree*. A directed graph is said to be a *chain* if it is an intree and an outtree at the same time. Figure 1.1 illustrates the relation between these different graph classes.

1.1.3 Polyhedra and Linear Programming

A commonly used approach to tackle combinatorial optimization problems is to encode the set of feasible solutions as a set of high-dimensional vectors and to transform the objective function to a linear function from the space of these vectors to the real line.

For TSP, we could, for instance, introduce a binary variable x_{ij} for every pair of cities i and j that indicates whether we go from city i directly to j ($x_{ij} = 1$) or not ($x_{ij} = 0$). The set of feasible tours can then be described as the set of all integer points in $\mathbb{R}^{n(n-1)/2}$ that satisfy certain constraints, see Example 1.4. The length of a tour is then the sum over all variables x_{ij} times the cost of traveling from i to j.

Example 1.4 (A Formulation for TSP)
The following formulation for TSP is due to Dantzig, Fulkerson and Johnson [35]. There is a one-to-one correspondence between optimal tours and optimal solutions of the following *integer program*:

$$\min \quad \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} c_{ij}\, x_{ij} \tag{1.3a}$$

$$\text{s.t.} \quad \sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} = 2 \qquad \forall\, i \in [n], \tag{1.3b}$$

$$\sum_{i \in S} \sum_{j \in S \setminus \{i\}} x_{ij} \leq |S| - 1 \qquad \forall\, S \subsetneq [n], \tag{1.3c}$$

$$x_{ij} \in \{0, 1\} \qquad \forall\, i, j \in [n],\ i \neq j. \tag{1.3d}$$

Here, c_{ij} indicates the cost of traveling from city i to j. Constraints (1.3b) ensure that we enter and leave every city exactly once, and constraints (1.3c) that the resulting solution does not contain any subtours, i.e., is indeed a proper tour.

The objects we obtain via this geometric interpretation are so-called *polyhedra*. The framework of optimizing linear functions over a polyhedron is known as *linear programming*. In the related field of *integer programming*, we want to optimize linear functions over all integral points in a polyhedron, as in Example 1.4. In this section, we recall the most important notions and results of the theory of integer and linear programming. The reader is referred to, e.g., [145, 131] for more details.

Polyhedra. We assume that a vector $x \in \mathbb{R}^n$ is a *column vector*, and denote the corresponding transposed *row vector* by x^T. The *scalar product* of two vectors $x, d \in \mathbb{R}^n$ is defined as $d^T x \coloneqq \sum_{q=1}^{n} d_q x_q$. For $d \in \mathbb{R}^n$ and $\gamma \in \mathbb{R}$, the set $\{x \in \mathbb{R}^n \mid d^T x = \gamma\}$ is called a *hyperplane* and $\{x \in \mathbb{R}^n \mid d^T x \leq \gamma\}$ is the corresponding *halfspace*. A set $P \subseteq \mathbb{R}^n$ is called a *polyhedron* if there is a matrix $D \in \mathbb{Z}^{m \times n}$ and a vector $b \in \mathbb{Z}^m$ such that $P = \{x \in \mathbb{R}^n \mid Dx \leq b\}$.[4] A bounded polyhedron is called a *polytope*. For $d \in \mathbb{R}^n$ and $\gamma \in \mathbb{R}$, we call $d^T x \leq \gamma$ a *constraint*, and we say the constraint is *valid for* P if $d^T y \leq \gamma$ for all $y \in P$. We say a constraint is *tight* at P if it is valid, and if it is satisfied with equality for some $y \in P$.

[4] $Dx \leq b$ means that each component of Dx is less or equal than the respective component of b.

Let $V = \{v_1, \dots, v_\ell\} \subseteq \mathbb{R}^n$ be a finite set of vectors. The *convex hull* of V is defined as the set $\text{conv}(V) \coloneqq \{\sum_{q=1}^{\ell} \lambda_q v_q \,|\, \lambda_1, \dots, \lambda_\ell \geq 0,\ \sum_{q=1}^{\ell} \lambda_q = 1\}$, and the *linear hull* of V is $\text{lin}(V) \coloneqq \{\sum_{q=1}^{\ell} \lambda_q v_q \,|\, \lambda_1, \dots, \lambda_\ell \in \mathbb{R}\}$. A set $X \subseteq \mathbb{R}^n$ is called an *affine subspace* if there is $x \in \mathbb{R}^n$ and a finite set $V \subseteq \mathbb{R}^n$ such that $X = \{x + v \,|\, v \in \text{lin}(V)\}$. The *dimension* of X is equal to the dimension of the linear subspace $\text{lin}(V)$. A set $Y \subseteq \mathbb{R}^n$ is called *convex* if $\lambda x + (1-\lambda) y \in Y$ for all $x, y \in Y$ and $\lambda \in [0;1]$. Note that polyhedra are convex. The dimension of a convex set $Y \subseteq \mathbb{R}^n$, denoted by $\dim(Y)$, is the dimension of an inclusion-minimal affine subspace that contains Y. A finite set $V = \{v_0, \dots, v_\ell\} \subseteq \mathbb{R}^n$ is called *affinely independent* if $\dim(\text{conv}(\{v_q - v_0 \,|\, q \in [\ell]\})) = \ell$.

Let P be a polyhedron, $H \subseteq \mathbb{R}^n$ a hyperplane, and let $H_\leq$ be the halfspace corresponding to H. If $P \subseteq H_\leq$ and $P \cap H \neq \emptyset$ then H is called *supporting hyperplane* at P, and $F = P \cap H$ is called a *face* of P. Note that F is a polyhedron. We call F a *vertex* of P if $\dim(F) = 0$, i.e., F is a singleton, and we call it a *facet* if $\dim(F) = \dim(P) - 1$. We say that the constraint $H_\leq$ *defines* the face F. Note that F is a facet of P if F contains an affinely independent set $V \subseteq F$ with $|V| = \dim(P)$.

Linear and Integer Programming. Let $P = \{x \in \mathbb{R}^n \,|\, Dx \leq b\}$ be a rational polyhedron and consider the linear function $c^T x$ for some $c \in \mathbb{Z}^n$. One can minimize $c^T x$ over P in polynomial time in the input size of P using the ellipsoid algorithm of Khachiyan [99, 49]. Note that maximizing $c^T x$ is equivalent to minimizing $-c^T x$, so we can, w.l.o.g., restrict to minimization problems. It is well-known that, if P is a polytope, there is an optimal solution of the *linear program*, or *LP*, $\min\{c^T x \,|\, x \in P\}$ that is a vertex of P. Grötschel, Lovász and Schrijver [70] showed that, even if the size of P is not polynomial in the input size of an optimization problem Π, one can still solve the linear program in polynomial time in the input size of Π, if one can solve the corresponding separation problem in polynomial time. The *separation problem* for an LP is, given a vector x, to decide whether $x \in P$ or find a valid constraint $d^T y \leq \gamma$ of P that is violated by x, i.e., such that $d^T y \leq \gamma < d^T x$ for all $y \in P$.

Definition 1.5 (Integer Program and LP Relaxation)
Let P be a rational polyhedron. Then $\min\{c^T x \,|\, x \in P \cap \mathbb{Z}^n\}$ is called an *integer program*, and $\min\{c^T x \,|\, x \in P\}$ its (*LP*) *relaxation*. A vector $x \in P$ is called *feasible* for the LP, and $x \in P \cap \mathbb{Z}^n$ is called *feasible* for the integer program. The set $\text{conv}(P \cap \mathbb{Z}^n)$ is called the *integer hull* of P.

The integer hull of a polyhedron is again a polyhedron, and solving the integer program $\min\{c^T x \,|\, x \in P \cap \mathbb{Z}^n\}$ is equivalent to solving the LP $\min\{c^T x \,|\, x \in \text{conv}(P \cap \mathbb{Z}^n)\}$ over the integer hull. Although we can optimize a linear function over a polyhedron P, it is NP-hard to optimize the function over $P \cap \mathbb{Z}^n$ in general. Even recognizing whether a polyhedron coincides with its integer hull, i.e., whether $P = \text{conv}(P \cap \mathbb{Z}^n)$, is NP-hard [133]. The optimal objective value of an integer program is not less than the optimal solution of its LP relaxation, i.e., $\min\{c^T x \,|\, x \in P \cap \mathbb{Z}^n\} \geq \min\{c^T x \,|\, x \in P\}$.

The ratio between the optimal objective value of an integer program and its relaxation is called the *integrality gap*.[5]

Definition 1.6 (Integrality Gap)
Let $\min\{c^T x \mid x \in P \cap \mathbb{Z}^n\}$ be an integer program and $\min\{c^T x \mid x \in P\}$ its LP relaxation. For $c \in \mathbb{Z}^n$, let x_c^* and $\overline{x}_c$ be optimal solutions of the integer program and its relaxation, respectively. The *integrality gap* of P is defined as the supremum over the ratios of the optimal objective values, $\sup_{c \in \mathbb{Z}^n} \frac{c^T x_c^*}{c^T \overline{x}_c}$.

There might be several LP relaxations for the same integer program, i.e., polytopes P and Q such that $P \cap \mathbb{Z}^n = Q \cap \mathbb{Z}^n$, but $P \neq Q$. If the integrality gap of P is less than the integrality gap of Q, we say that P is *stronger* than Q. Typically, we want to find LP relaxations for combinatorial optimization problems that are as strong as possible, but can still be solved in polynomial time in the input size of the problem.

A common approach to design approximation algorithms, which we use in Chapters 4 and 5, is to come up with a strong LP relaxation for the problem and bound the cost of the solution returned by the algorithm by the cost of an optimal LP solution. That is, given a problem Π, find $D \in \mathbb{Z}^{m \times n}$ and $b \in \mathbb{Z}^m$ and set $P = \{x \in \mathbb{R}^n \mid Dx \leq b\}$ such that there is a one-to-one correspondence between integer points in $P \cap \mathbb{Z}^n$ and feasible solutions of Π. If the algorithm solves the LP over P to obtain a feasible integer solution, e.g., through rounding, then one has to take care that the LP can be solved in polynomial time. Such approximation algorithms are called *LP based*. Note that the integrality gap gives a lower bound on the approximation ratio of an LP based approximation algorithm.

Example 1.7 (Example 1.4 continued)
An LP relaxation of the integer program (1.3) is obtained by relaxing the binary constraints (1.3d) to $0 \leq x_{ij} \leq 1$. This LP relaxation is known as the Held-Karp relaxation [80]. One can show that the integrality gap of this relaxation is at least $\frac{4}{3}$ [164]. Also, it is possible to efficiently separate constraints (1.3c) [35], i.e., we can solve the LP relaxation in polynomial time in the size of the TSP instance.

1.1.4 Scheduling

Since we consider two different main scheduling settings in this thesis (see Section 1.3), we define in this section the notation for the main part of the thesis (Chapters 2 to 4) only. The terminology and notation for the scheduling problem presented in the last chapter is deferred to Section 5.1. We trust the reader will find this presentation more natural rather than confusing.

[5] Technically, one has to be careful about the existence of optimal solutions and the objective values being equal to zero or of different signs. However, since we consider integer programs and LP relaxations of specific combinatorial optimization problems only, we can ignore such pathological examples. That is, the integrality gap is well-defined and greater or equal than 1.

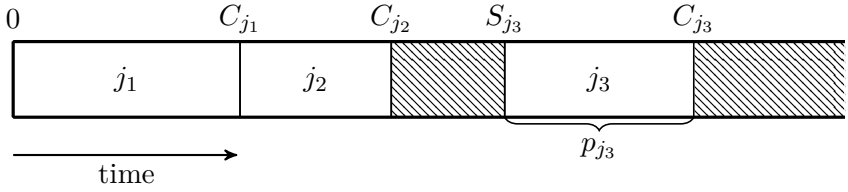

Fig. 1.2: A non-preemptive single-machine schedule $j_1 \to j_2 \to j_3$. In sketches of schedules, shaded areas correspond to idle time, and jobs are depicted as rectangles with length corresponding to their processing time. Note that the machine is not idle at time C_{j_1} because the next job immediately starts processing, i.e., $C_{j_1} = S_{j_2}$.

Schedules. Let $n, m \in \mathbb{N}$. Let N be a set of n jobs and m be the number of machines. We consider the setting of m parallel identical machines, i.e., all machines are of equal speed and run in parallel. We call the corresponding scheduling problem a *single-machine problem* if $m = 1$, and a *parallel-machine problem* if $m > 1$. Each job $j \in N$ is associated with a processing time $p_j \geq 0$, a weight $w_j \geq 0$ and a release date $r_j \geq 0$. For a subset of jobs $S \subseteq N$, we abbreviate $p(S) \coloneqq \sum_{j \in S} p_j$ and $w(S) \coloneqq \sum_{j \in S} w_j$. By scaling the input data suitably, we may assume that all data are integral.

Definition 1.8 (Schedule)
A *schedule* is an assignment of the jobs in N to the machines such that

(i) each job j is processed by a machine for p_j disjoint units of time,
(ii) no job starts before its release date, and
(iii) each machine processes only one job at a time.

Depending on the problem definition, jobs may be allowed to preempt and continue on another machine (*preemptive scheduling*) or not (*non-preemptive scheduling*).

The *start time* and *completion time* of job $j \in N$ is denoted by S_j and C_j, respectively. Note that $C_j \geq S_j + p_j$, and equality holds if job $j \in N$ is not preempted. If job i directly precedes j on the same machine, we denote this by $i \to j$. A machine is *idle in the interval* $[s; t]$ if no job is processed by the machine at any point in time $s < t' < t$. We call a machine *idle at time* t if there is a sufficiently small $\varepsilon > 0$ such that the machine is idle in the interval $[t; t + \varepsilon]$. An *idle interval* is an interval where some machine is idle. Figure 1.2 illustrates a feasible single-machine schedule of three jobs.

Additionally, we consider precedence constraints on the set of jobs in form of a directed graph $G = (N, E)$, which we refer to as the *precedence graph*. We call $i \in N$ a *predecessor* of job $j \in N$ if $(i, j) \in E$. Feasibility of a schedule depends on the specific type of precedence relation. Typically, the literature focuses on what we call *AND-precedence constraints*, see, e.g., [68, 111, 27]. In this case, any job $j \in N$ requires *all* its predecessors to be completed before it can start, i.e., $C_i \leq S_j$ for all $(i, j) \in E$.

For so-called *AND/OR-precedence constraints*, the predecessors of a job are partitioned into *AND-predecessors* and *OR-predecessors*, see, e.g. [57, 43, 125]. In this setting, a job j requires *all* its AND-predecessors and *at least one* of its OR-predecessors to be completed before it can start. The case where all predecessors are OR-predecessors is called *scheduling with OR-precedence constraints* and constitutes the main part of this thesis. We elaborate on the different types of precedence constraints in Section 1.2.

Scheduling Environments. Graham et al. [68] introduced a three-field notation $\alpha \,|\, \beta \,|\, \gamma$ to classify scheduling problems. Since then, other authors have extended the original three-field notation and incorporated new settings. In this section, we briefly mention the settings that are considered in this thesis.

The first parameter, α, indicates the machine environment. We mainly focus on $\alpha \in \{1, P, Pm\}$ where the specific values indicate

1	: single-machine environment ($m = 1$),
P	: parallel-identical-machine environment ($m > 1$),
Pm	: parallel-identical-machine environment ($m > 1$) with fixed m.

The second parameter, β, describes the properties of the jobs. If $\beta = \emptyset$, then we assume the *default setting* where jobs must not be preempted, all release dates are equal to zero ($r_j = 0$ for all $j \in N$) and there are no precedence constraints ($E = \emptyset$ in the precedence graph). Else, β indicates the deviations from this default setting. The most important values that β can attain in our settings are

r_j	: jobs have non-trivial release dates,
$pmtn$	: jobs may be preempted and continue on a different machine,
$prec$	: jobs are subject to AND-precedence constraints,
$or\text{-}prec$	: jobs are subject to OR-precedence constraints,
$ao\text{-}prec$	: jobs are subject to AND/OR-precedence constraints,
$p_j = 1$	: jobs have unit-processing time,
$p_j \in \{0, 1\}$	: jobs have 0/1 processing time.

Note that β can be a subset of these values. If the precedence graph is, e.g., an outtree, we denote this by, e.g., $prec = outtree$. If the precedence graph is an outforest we can transform it to an outtree by adding a job with zero processing time and zero weight and introducing an arc from this job to all roots of the outforest. It can be easily verified that adding such a job does not alter the optimal solution. Similarly, we can transform any inforest to an intree.

The last field, γ, describes the objective function of the problem. The two objective functions we consider are minimizing the *sum of weighted completion times* or *total weighted completion time*, $\sum_{j \in N} w_j C_j$, and minimizing the *makespan*, $C_{\max} := \max_{j \in N} C_j$.

The objective functions are encoded via γ by the values

$\sum w_j C_j$: minimize the sum of weighted completion times,
$\sum C_j$: minimize the sum of completion times ($w_j = 1$ for all $j \in N$),
$C_{\max}$: minimize the makespan.

Instead of minimizing the total weighted completion time, $\sum w_j C_j$, one could also minimize the *average completion time*, $\frac{1}{n}\sum w_j C_j$. Note that this is indeed equivalent.

1.2 Min-Sum Set Cover and Scheduling with OR-Precedence Constraints

The two major problems that are considered in this thesis are called *min-sum set cover* and *OR-scheduling*. In this section, we formally define the two problems as well as some generalizations and discuss their connection. All of these problems are NP-hard, as discussed in the main part of the thesis.

1.2.1 Min-Sum Set Cover and Some Generalizations

The following problem was first formulated by Feige, Lovász and Tetali [44]. Let U be a finite set of elements and $\mathcal{R} \subseteq 2^U$ a collection of subsets. For a given linear ordering $\pi : U \to [|U|]$ of the elements in U, we define the *covering time* of $R \in \mathcal{R}$ as the first point in time when an element in R appears in the linear ordering, $\pi(R) \coloneqq \min_{e \in R} \pi(e)$.

Definition 1.9 (Min-Sum Set Cover (MSSC))
Let U be a finite set of elements and $\mathcal{R} \subseteq 2^U$. The task is to find a linear ordering $\pi : U \to [|U|]$ that minimizes the sum of covering times, $\sum_{R \in \mathcal{R}} \pi(R)$.

Note that MSSC can be seen as a min-sum variant of the well-known Set Cover or the equivalent Hitting Set problem, which are NP-complete [98]. A natural generalization of MSSC is to introduce *covering requirements* on the sets in $\mathcal{R}$. In this case, every set $R \in \mathcal{R}$ is associated with an integer $\kappa(R) \in [|R|]$, and the covering time of R is then the first point in time when $\kappa(R)$ of its elements have appeared in the linear ordering. That is, the covering time of $R \in \mathcal{R}$ is

$$\pi(R) \coloneqq \min \{\ell \in [|U|] \,|\, |\{\pi^{-1}(1), \dots, \pi^{-1}(\ell)\} \cap R| = \kappa(R)\}. \tag{1.4}$$

This problem is known as *generalized min-sum set cover* and was introduced in a different but equivalent way under the term *multiple intents re-ranking* in [10] in the context of web page rankings.[6]

[6]The term *generalized min-sum set cover* was coined by Bansal, Gupta and Krishnaswamy [14].

Definition 1.10 (Generalized Min-Sum Set Cover (GMSSC))
Let U be a finite set of elements and $\mathcal{R} \subseteq 2^U$ with covering requirements $\kappa : \mathcal{R} \to \mathbb{N}$. The task is to find a linear ordering $\pi : U \to [|U|]$ that minimizes the sum of covering times, $\sum_{R \in \mathcal{R}} \pi(R)$.

Note that MSSC is the special case of GMSSC with unit covering requirements. The case where all covering requirements are equal to the cardinality of the sets, i.e., $\kappa(R) = |R|$ for all $R \in \mathcal{R}$, is known as *minimum latency set cover* (MLSC) and was studied in [79]. Yet another important generalization of MSSC, which was proposed in [128] and studied in [122], is to incorporate a partial order $\prec$ on the elements in U.[7] That is, any feasible linear ordering $\pi : U \to [|U|]$ must satisfy $\pi(e) < \pi(e')$ whenever $e \prec e'$. This problem is known as *precedence-constrained min-sum set cover*.

Definition 1.11 (Precedence-Constrained MSSC (prec-MSSC))
Let U be a finite set of elements, $\prec \subseteq U \times U$ a partial order on U, and $\mathcal{R} \subseteq 2^U$. The task is to find a feasible linear ordering $\pi : U \to [|U|]$, i.e., $e \prec e'$ implies $\pi(e) < \pi(e')$, that minimizes the sum of covering times, $\sum_{R \in \mathcal{R}} \pi(R)$.

We can readily encode instances of these set covering problems via directed graphs. Let $H = (U \cup \mathcal{R}, E)$ be a digraph with one node for every element in U and every set in $\mathcal{R}$. We introduce an arc $(e, R) \in E$ from an element e to a set R if $e \in R$. The resulting digraph H is called the *covering graph* of the instance and is bipartite. We discuss these min-sum covering problems and other related problems in more detail in Chapters 3 and 4.

Example 1.12 (Min-Sum Set Cover)
Consider the following instance of MSSC with $U = [6]$ and subsets

$$\mathcal{R} = \{\{1,2\},\{2,3\},\{2,4\},\{1,4,5\},\{3,5,6\},\{4,5,6\}\}. \tag{1.5}$$

Figure 1.3 illustrates the instance and its covering graph. An optimal ordering of the elements is $2 \to 5 \to 1 \to 3 \to 4 \to 6$ with an objective value of 9.

1.2.2 Scheduling with OR-Precedence Constraints

Consider an instance of MSSC represented by its covering graph $H = (U \cup \mathcal{R}, E)$. If we interpret every node of H as a job, we can view MSSC as a scheduling problem: For each element $e \in U$, we introduce a job j_e with processing time $p_{j_e} = 1$ and weight $w_{j_e} = 0$, and for each set $R \in \mathcal{R}$, we introduce a job j_R with processing time $p_{j_R} = 0$ and weight $w_{j_R} = 1$. The arc set of H are OR-precedence constraints among the jobs.

[7] Any reader not familiar with partial orders is referred to [161] for more details.

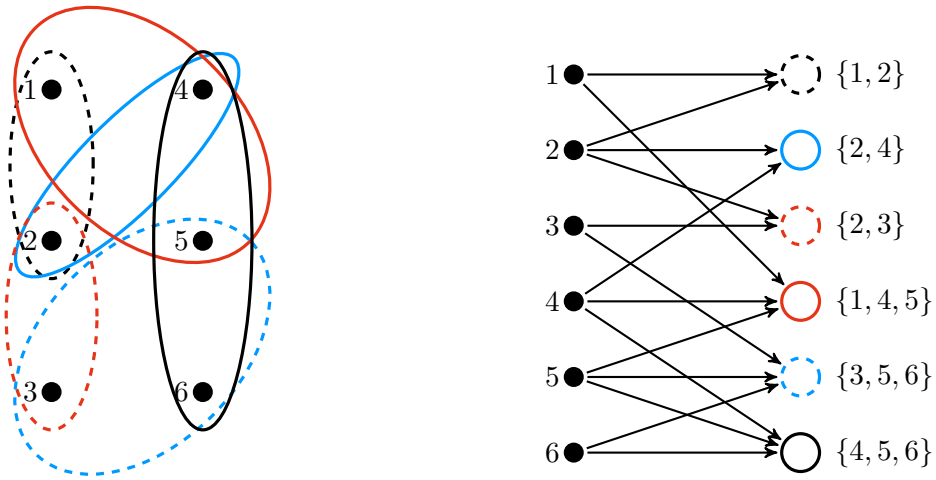

Fig. 1.3: The MSSC instance in Example 1.12 represented by elements and sets (left) and the corresponding covering graph (right). The sets are depicted as colored dashed and solid ellipses. For simplicity, the nodes in the covering graph corresponding to the sets are depicted in the same style as the respective sets on the left.

That is, every set-job j_R requires at least one of its predecessors in $\{j_e \mid e \in R\}$ to be completed before it can start processing.

Then, every single-machine schedule of the element-jobs in $\{j_e \mid e \in U\}$ corresponds to a linear ordering of the elements in U, and vice versa. Furthermore, minimizing the sum of covering times, $\sum_{R \in \mathcal{R}} \pi(R)$, is equivalent to minimizing the sum of weighted completion times, $\sum_j w_j C_j$, subject to the OR-precedence constraints given by H. Hence, MSSC can be interpreted as a single-machine scheduling problem with 0/1 processing times and 0/1 weights. In this section, we define *scheduling with OR-precedence constraints*, which constitutes the main part of this thesis.

OR-Scheduling. Recall the scheduling notation from Section 1.1.4. Let N be a set of n jobs with processing times $p_j \in \mathbb{N}_0$, weights $w_j \in \mathbb{N}_0$ and release dates $r_j \in \mathbb{N}_0$ for all $j \in N$. Let $G = (N, E_\vee)$ be a precedence graph on the jobs in N. For $j \in N$, we denote the set of its *predecessors* by $\mathcal{P}(j) \coloneqq \{i \in N \mid (i, j) \in E_\vee\}$.

Definition 1.13 (OR-Scheduling)
Let N be a set of jobs and $G = (N, E_\vee)$ a precedence graph. A schedule is said to be *feasible with respect to the OR-precedence constraints of* G if

(i) it is feasible according to Definition 1.8, and
(ii) if $\mathcal{P}(j) \neq \emptyset$ for $j \in N$, then there is $i \in \mathcal{P}(j)$ that precedes j, i.e., $C_i \leq S_j$.

A job $j \in N$ is called *available* at time $t \geq 0$ if $t \geq r_j$ and, unless $\mathcal{P}(j) = \emptyset$, there is $i \in \mathcal{P}(j)$ with $C_i \leq t$. We call an instance of OR-scheduling *bipartite* if the precedence graph G is bipartite. Recall that interpreting MSSC as a scheduling problem yields a bipartite OR-scheduling instance to minimize the sum of weighted completion times.

The main variants of OR-scheduling considered in this thesis are *makespan minimization with and without preemption* and *minimizing the sum of weighted completion times on a single machine*. In an extension of the three-field notation of Graham et al. [68] and [93], we are concerned with the following scheduling problems: $P \,|\, r_j,\, or\text{-}prec \,|\, C_{\max}$, $P \,|\, r_j,\, pmtn,\, or\text{-}prec \,|\, C_{\max}$ (Chapter 2) and $1 \,|\, or\text{-}prec \,|\, \sum w_j C_j$ (Chapters 3 and 4). In contrast to OR-scheduling as defined in Definition 1.13, the standard precedence constraints considered in the literature, where a job requires *all* its predecessors to be completed, are referred to as *AND-precedence constrained scheduling* or *AND-scheduling*.[8] The arc set in the corresponding precedence graph is then denoted by $E_\wedge$.

In some statements and proofs it is convenient to assume that there is a unique job without predecessors. We can, w.l.o.g., add such a job with zero processing time and zero weight, and introduce an arc to all jobs $j \in N$ with $\mathcal{P}(j) = \emptyset$ in the precedence graph without changing the instance. We call the unique job without predecessors *initial* and denote it by j^{in}. The corresponding precedence graph is $G^{\text{in}} := (N \cup \{j^{\text{in}}\}, E_\vee \cup \{(j^{\text{in}}, j) \,|\, \mathcal{P}(j) = \emptyset\})$. An importation notion for OR-scheduling is the notion of a *feasible starting set*.

Definition 1.14 (Feasible Starting Sets)
Let N be a set of jobs and $G = (N, E_\vee)$ a precedence graph. A set $S \subseteq N$ is called a *feasible starting set* if all jobs in S are reachable from j^{in} in the induced subgraph $G^{\text{in}}[S \cup \{j^{\text{in}}\}]$. The set of feasible starting sets is denoted by $\mathcal{FS}$.

If the precedence graph G is acyclic, we can characterize the set of feasible starting sets equivalently as $\mathcal{FS} = \{S \subseteq N \,|\, j \in S \text{ and } \mathcal{P}(j) \neq \emptyset \Rightarrow \mathcal{P}(j) \cap S \neq \emptyset\}$. In some sense, feasible starting sets can be seen as the counterpart of *ideals* of a partial order (see [161]) in AND-scheduling, i.e., a subset of jobs that is closed under precedence constraints. Note that $N \in \mathcal{FS}$ if and only if all jobs in N are reachable from j^{in} in G^{in}. Whether or not all jobs are reachable in G^{in} can be checked by a graph traversing algorithm that starts at j^{in}, see Section 1.1.2. So, we can check whether a given instance of OR-scheduling has a feasible solution in polynomial time. Henceforth, we assume that the instances we consider do have a feasible solution, i.e., $N \in \mathcal{FS}$. Note that, for OR-scheduling, the precedence graph may contain a cycle, whereas an instance of AND-scheduling is feasible if and only if the precedence graph is acyclic.

Example 1.15 (OR-Scheduling and AND-Scheduling)
Let $N = \{i, j, k, k'\}$ be a set of jobs of unit processing time and consider the precedence graph $G = (N, E)$ with $E = \{(i,k), (i,k'), (j,k), (k,k')\}$. Figure 1.4 (left) illustrates the precedence graph together with the initial job j^{in}.

[8] That is, those scheduling problems with $prec \in \beta$ in the notation of [68], see Section 1.1.4.

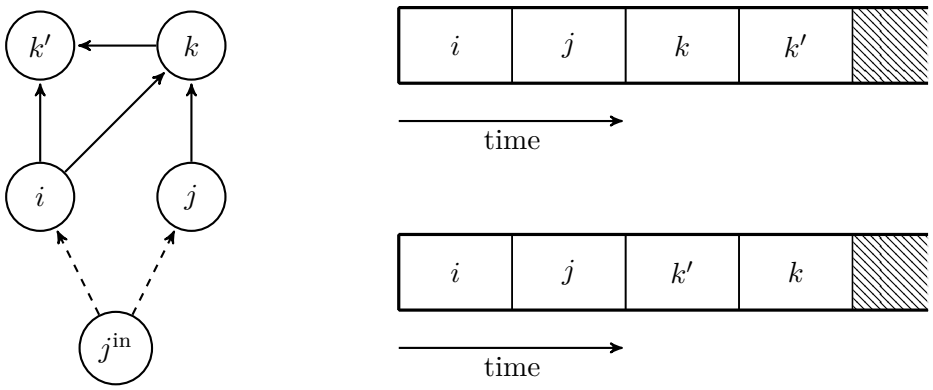

Fig. 1.4: The precedence graph of the instance in Example 1.15 (left) and two schedules (right). The arcs (j^{in}, i) and (j^{in}, j) from the initial job to the jobs without predecessors are depicted as dashed arcs. The schedule on the top right is feasible for both AND-constraints, i.e., $E = E_\wedge$, and OR-constraints, i.e., $E = E_\vee$. The schedule on the bottom right is only feasible for OR-constraints, since $k' \to k$.

The schedule $i \to j \to k \to k'$ (Figure 1.4 top right) is feasible for AND-precedence constraints, i.e., for $E = E_\wedge$, and OR-precedence constraints, i.e., $E = E_\vee$. The schedule $i \to j \to k' \to k$ (Figure 1.4 bottom right) is only feasible for $E = E_\vee$, since k' precedes k, which is not feasible for AND-constraints. For OR-precedence constraints, i.e., $E = E_\vee$, the set of feasible starting sets $\mathcal{FS}$ is

$$\{\{i\}, \{j\}, \{i,j\}, \{i,k\}, \{i,k'\}, \{j,k\}, \{i,j,k\}, \{i,j,k'\}, \{i,k,k'\}, \{j,k,k'\}, N\}. \quad (1.6)$$

Note that, e.g., $\{j, k'\} \notin \mathcal{FS}$, since there is no path from j^{in} to k' in $G^{\text{in}}[\{j^{\text{in}}, j, k'\}]$.

AND/OR-Scheduling. A mutual generalization of AND-scheduling and OR-scheduling is to consider jobs that are subject to both AND-precedence constraints and OR-precedence constraints. In this setting, the arc set of the precedence graph can be partitioned into $E = E_\wedge \dot\cup E_\vee$, where $(i,j) \in E_\wedge$ indicates that i is an AND-predecessor of j, and $(i,j) \in E_\vee$ means that i is an OR-predecessor of j. Again, we denote the set of *OR-predecessors* of a job $j \in N$ by $\mathcal{P}(j) := \{i \in N \mid (i,j) \in E_\vee\}$.

Definition 1.16 (AND/OR-Scheduling)
Let N be a set of jobs and $G = (N, E_\wedge \dot\cup E_\vee)$ a precedence graph. A schedule is said to be *feasible with respect to the AND/OR-precedence constraints of* G if

- (i) it is feasible according to Definition 1.8,
- (ii) $C_i \le S_j$ for all $(i,j) \in E_\wedge$, and
- (iii) if $\mathcal{P}(j) \neq \emptyset$ for $j \in N$, then there is $i \in \mathcal{P}(j)$ such that $C_i \le S_j$.

For AND/OR-scheduling, a job $j \in N$ is called *available* at time $t \geq 0$ if $t \geq r_j$, $t \geq \max\{C_i \mid (i,j) \in E_\wedge\}$ and, unless $\mathcal{P}(j) = \emptyset$, there is $i \in \mathcal{P}(j)$ with $C_i \leq t$.[9] Obviously, AND-scheduling and OR-scheduling are special cases of Definition 1.16 where $E_\vee = \emptyset$ and $E_\wedge = \emptyset$, respectively. We only focus on minimizing the sum of weighted completion times on a single machine subject to certain types of AND/OR-precedence constraints. The interested reader is referred to, e.g., [57, 43, 125] and references therein for more details and other AND/OR-scheduling problems. The following problem is denoted by $1 \mid \textit{ao-prec} = A \dot{\vee} B \mid \sum w_j C_j$ in an extension of the three-field notation of [68] and [43], and is studied in Section 4.3.1.

Definition 1.17 (Bipartite AND/OR-Scheduling)
Let $N = A \dot{\cup} B$ be a set of jobs and $G = (N, E_\wedge \dot{\cup} E_\vee)$ be a precedence graph with $E_\wedge \subseteq (A \times A) \cup (B \times B)$ and $E_\vee \subseteq A \times B$. The task is to find a feasible single-machine schedule according to Definition 1.16 that minimizes the sum of weighted completion times, $\sum_{j \in N} w_j C_j$.

One can check in polynomial time whether a given instance of AND/OR-scheduling is feasible [125]. For bipartite AND/OR-scheduling, it is not hard to see that an instance is feasible if and only if the subgraph of G on $E_\wedge$, i.e., the restriction to the AND-precedence constraints, is acyclic. Therefore, we also assume that all instances of bipartite AND/OR-scheduling that are considered have a feasible solution. To distinguish between AND-scheduling, OR-scheduling and AND/OR-scheduling, we denote the arc set of the precedence graph by $E_\wedge$, $E_\vee$ and $E_\wedge \dot{\cup} E_\vee$, respectively.

Min-Sum Set Cover and Scheduling. Recall that we can express min-sum set cover (Definition 1.9) as a bipartite OR-scheduling instance where all processing times and weights are 0/1. In a similar way, bipartite AND/OR-scheduling, $1 \mid \textit{ao-prec} = A \dot{\vee} B \mid \sum w_j C_j$ (Definition 1.17), generalizes precedence-constrained min-sum set cover (Definition 1.11): We introduce a job $j_e \in A$ with $p_{j_e} = 1$ and $w_{j_e} = 0$ for every element $e \in U$, and a job $j_R \in B$ with $p_{j_R} = 0$ and $w_{j_R} = 1$ for every set $R \in \mathcal{R}$. Further, we let $E_\wedge = \{(j_e, j_{e'}) \mid e \prec e'\} \subseteq A \times A$ and $E_\vee = \{(j_e, j_R) \mid e \in R\} \subseteq A \times B$. Then, as for min-sum set cover and bipartite OR-scheduling, finding a feasible single-machine schedule that minimizes the sum of weighted completion times is equivalent to finding a feasible linear ordering that minimizes the sum of the covering times.

Single-machine scheduling with AND-precedence constraints to minimize the sum of weighted completion times is not only a special case of AND/OR-scheduling (Definition 1.16), but also of bipartite AND/OR-scheduling (Definition 1.17), where $E_\vee = \emptyset$. Woeginger [166] showed that every instance of $1 \mid \textit{prec} \mid \sum w_j C_j$ can be transformed to a bipartite AND-scheduling instance with $N = A \dot{\cup} B$ and $E_\wedge \subseteq A \times B$ of the same

[9] We set the maximum over the empty set to $\max \emptyset := -\infty$.

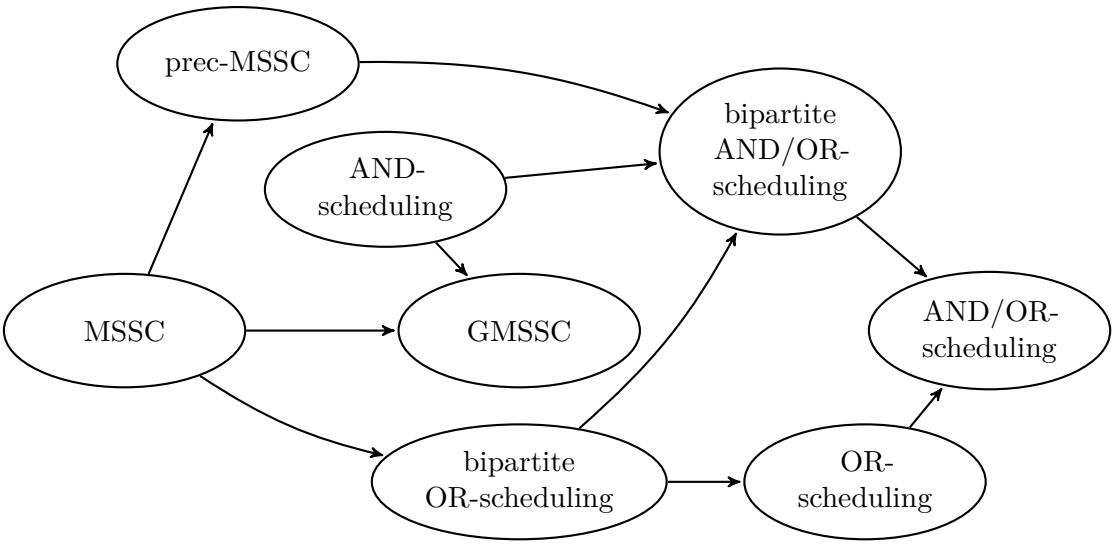

Fig. 1.5: Overview of related problems defined in Section 1.2. All scheduling problems are single-machine problems to minimize the sum of weighted completion times, $\sum w_j C_j$. An arrow from problem Π_1 to Π_2 indicates that Π_2 generalizes Π_1.

approximability threshold, where all jobs in A have unit processing time and zero weight, and all jobs in B have zero processing time and unit weight. The precedence graph of this bipartite instance can be viewed as the covering graph of an instance of generalized min-sum set cover with an element for every job in A, and a set for every job in B. Hence, $1\,|\,prec\,|\,\sum w_j C_j$ is equivalent to minimum latency set cover, which is the special case of generalized min-sum set cover, where all covering requirements are maximal, i.e., $\kappa(R) = |R|$ for all $R \in \mathcal{R}$. We can also interpret generalized min-sum set cover in general as a bipartite scheduling problem with certain precedence constraints where every job requires a certain number of its predecessors to be completed before it can start. This extension of generalized min-sum set cover is discussed in more detail in Sections 3.4 and 4.3.2.

Figure 1.5 illustrates the connection of the min-sum covering problems defined in Section 1.2.1 and the single-machine scheduling problems to minimize the sum of weighted completion times described above.

1.3 Overview of the Thesis and Main Results

The main part of this thesis comprises four chapters with Chapter 5 addressing a different scheduling model, and which is written to be (almost) stand-alone except for some general notions in Section 1.1. Chapters 2 to 4, which are concerned with different variants of min-sum set cover and scheduling with OR-precedence constraints, are subdivided in a similar way.

First, we give a short overview of the respective chapter and our main results. We discuss related work and connect the related models and results from the literature to our results and the problems considered in this work. In the preliminaries section, we recall related algorithms, methods and definitions mostly from AND-scheduling and introduce some important notions, if necessary. Afterwards, we state our main results, and conclude each chapter with an open problems section, which is intended as a starting point for future work. We elaborate on the structure and content of the respective main parts of the chapters in the following paragraphs.

In Chapter 2, we study the makespan objective in a parallel machine environment subject to OR-precedence constraints. This chapter largely coincides with work that is available online in [74]. We show that the well-known *list scheduling* algorithm, which is due to Graham [66], is a 2-approximation algorithm for $P \,|\, r_j,\ \textit{or-prec} \,|\, C_{\max}$ (Theorem 2.8) in Section 2.3.1. In Section 2.3.2, we propose a reduction from the NP-hard Vertex Cover problem and conclude that the minimum makespan cannot be approximated within a factor of $\frac{4}{3} - \varepsilon$, unless P = NP (Theorem 2.11). Finally, in Section 2.4, we use the concept of *minimal chains*, which we introduce in Section 2.2.2, and an algorithm of Lawler [109] for a related AND-scheduling problem to derive a polynomial-time algorithm for the preemptive variant, i.e., for $P \,|\, r_j,\ \textit{pmtn},\ \textit{or-prec} \,|\, C_{\max}$ (Theorem 2.18).

Chapter 3 deals with the connection of variants of (generalized) min-sum set cover and OR-scheduling to minimize the sum of weighted completion times. The results in this chapter are also contained in work that is in preparation [75, 76, 77]. The algorithms presented in this chapter are purely combinatorial, i.e., do not require solving a linear program. In Section 3.3, we observe that bipartite OR-scheduling is already strongly NP-hard, even for the simplest non-trivial processing times and weights structure (Theorem 3.6). We then show, in Section 3.4, that min-sum set cover and special cases of generalized min-sum set cover are solvable in polynomial time if we restrict to *laminar* sets (Theorem 3.11). We also propose a 2-approximation algorithm for laminar generalized min-sum set cover (Theorem 3.13 and Corollary 3.14). In Section 3.5, we generalize a histogram argument of [45] for min-sum set cover to more general scheduling problems and derive a general framework to obtain 4η-approximation algorithms (Theorem 3.15), if an η-approximation oracle is available for computing a *density-maximizing* feasible starting set. We present the framework in Section 3.5.1, and apply it to OR-scheduling problems to obtain 4-approximate solutions if the precedence graph is bipartite (Theorem 3.17) or in the form of an intree (Theorem 3.19) in Section 3.5.2.

Chapter 4 is devoted to linear programming relaxations for OR-scheduling problems with the total weighted completion time objective. An extended abstract of the results in this chapter was published in [78], and a journal version of this work is available in [77]. We first review the three classical LP relaxations in the literature of AND-scheduling in Section 4.2, and discuss their OR-scheduling counterparts in Sections 4.3

to 4.5. We present approximation algorithms based on *time-indexed* LPs for bipartite AND/OR-scheduling (Theorem 4.4) and a special case of generalized min-sum set cover (Theorem 4.10) with approximation guarantees of $2 \max_{b \in B} |\mathcal{P}(b)|$ and 4 in Sections 4.3.1 and 4.3.2, respectively. In Section 4.4, we discuss a formulation in *linear ordering variables*, derive facet-defining inequalities for the integer hull (Theorems 4.13 and 4.17), and show that the integrality gap of this formulation grows linear in the number of jobs, even if we add these facet-defining constraints. Lastly, we present a natural extension of the *completion time formulation* to OR-precedence constraints by generalizing the well-known *parallel inequalities* of Wolsey [167] and Queyranne [139] using a generalization of *minimal chains* (Theorem 4.23). We prove that this formulation also exhibits an unbounded gap between optimal LP solution and optimum feasible schedule.

Finally, in Chapter 5, we consider the *preemptive concurrent open shop* setting, which is a slightly different setting than in the previous chapters. We first introduce the model and discuss related work in Section 5.1, and then present a 2-approximation algorithm for the variant with non-trivial release dates (Theorem 5.3) in Section 5.2. The algorithm is based on a linear program in completion time variables, which is introduced in Section 5.2.1, and uses a preemptive list scheduling algorithm of [71], which is presented in Section 5.2.2. This chapter, too, is concluded with some open problems.

Chapter 2

Makespan Minimization with OR-Precedence Constraints

In this chapter, we focus on OR-scheduling on parallel identical machines to minimize the completion time of the last job. This chapter largely coincides with work that is available online in [74].

2.1 Related Work and Our Results

Recall the notations of Sections 1.1.4 and 1.2.2. In this chapter, we study the following problems, denoted by $P \,|\, r_j,\, or\text{-}prec \,|\, C_{\max}$ (non-preemptive variant) and $P \,|\, r_j,\, pmtn,\, or\text{-}prec \,|\, C_{\max}$ (preemptive variant) in an extension of the three-field notation of Graham et al. [68] and [93].

Definition 2.1 (OR-Scheduling to Minimize the Makespan)
Let $n, m \in \mathbb{N}$, $m \geq 2$, and let N be a set of n jobs and $G = (N, E_\vee)$ be a precedence graph. The task is to find a feasible schedule on m machines according to Definition 1.13 that minimizes the makespan, $C_{\max} = \max\{C_j \,|\, j \in N\}$.

We recall some preliminaries from related problems and introduce some terminology in Section 2.2. In Section 2.3, we show that a standard *list scheduling* algorithm achieves an approximation guarantee of 2 for $P \,|\, r_j,\, or\text{-}prec \,|\, C_{\max}$, and that obtaining a $\left(\frac{4}{3} - \varepsilon\right)$-approximation for $P \,|\, or\text{-}prec \,|\, C_{\max}$ is NP-hard for any $\varepsilon > 0$. Finally, we show how we can use algorithms for the related AND-scheduling problems on outtrees to obtain polynomial-time algorithms for $P \,|\, r_j,\, pmtn,\, or\text{-}prec \,|\, C_{\max}$ and $P \,|\, r_j,\, or\text{-}prec,\, p_j = 1 \,|\, C_{\max}$ in Section 2.4. We start out with some related work.

Non-Preemptive Scheduling. Garey and Johnson [52] proved that the non-preemptive variant of makespan minimization is already strongly NP-hard in the absence of precedence constraints and release dates. The problem remains NP-hard, even if the number of machines is fixed to $m = 2$ [113]. The reductions are from 3-Partition, which is strongly NP-complete, to $P \,||\, C_{\max}$ and from Partition to $P2 \,||\, C_{\max}$, respectively.

In his seminal paper, Graham [66] showed that a simple algorithm called *List Scheduling* achieves an approximation guarantee of 2 for $P\,|\,|\,C_{\max}$:

Consider the jobs in any arbitrary order. Whenever a machine is idle, execute the next available job in the order on this machine. If no jobs are available, then wait until the next point in time when a job becomes available.

We discuss this algorithm and its performance guarantee in more detail in the next section. If the jobs are sorted in order of non-increasing processing times, then List Scheduling is a $\frac{4}{3}$-approximation [67]. In the same paper [67], Graham also provided a PTAS for $Pm\,|\,|\,C_{\max}$, i.e., if the number of machines is fixed. Sahni [144] presented an FPTAS for $Pm\,|\,|\,C_{\max}$, and Hochbaum and Shmoys [84] gave the first PTAS for $P\,|\,|\,C_{\max}$. The algorithm in [84] was improved in running time to the currently best-known by Jansen [92]. Note that a PTAS is indeed the best algorithm for $P\,|\,|\,C_{\max}$ to be expected, since the existence of an FPTAS for a strongly NP-hard problem would imply P = NP [52]. For non-trivial release dates, List Scheduling with an arbitrary job order is a 2-approximation as observed in [72], and it is $\frac{3}{2}$-approximate if the jobs are sorted in order of non-increasing processing times [28]. Hall and Shmoys [72] also provided a PTAS for $P\,|\,r_j\,|\,C_{\max}$.

Minimizing the makespan with AND-precedence constraints is strongly NP-hard, even if the number of machines is fixed to $m = 2$ and the precedence graph consists of disjoint chains [40]. That is, the problem $P2\,|\,prec = chains\,|\,C_{\max}$ is already strongly NP-hard by a reduction from 3-Partition. List Scheduling is still 2-approximate for $P\,|\,prec\,|\,C_{\max}$ if the order of the jobs is consistent with the AND-precedence constraints [66]. The approximation factor can also be preserved for $P\,|\,r_j,\,prec\,|\,C_{\max}$ [72].

AND/OR-scheduling with an acyclic precedence graph and trivial release dates also admits a 2-approximation algorithm [57]. Erlebach, Kääb and Möhring [43] showed that the assumption on the precedence graph is not necessary, and presented a 2-approximation for $P\,|\,ao\text{-}prec\,|\,C_{\max}$. Both algorithms first transform the instance to an AND-scheduling instance by fixing a predecessor of the OR-precedence constraints for every job j with $\mathcal{P}(j) \neq \emptyset$. Then, they solve the resulting instance with AND-precedence constraints using List Scheduling. In Section 2.3.1, we show that the makespan of *every* feasible schedule without unnecessary idle time on the machines is at most twice the optimal makespan, even if non-trivial release dates are involved.

Preemptive Scheduling and Unit Processing Times. If preemption is allowed, the algorithm of McNaughton [123] computes an optimal schedule in the absence of release dates and precedence constraints, i.e., $P\,|\,pmtn\,|\,C_{\max}$ is solvable in polynomial time. Minimizing the makespan with AND-precedence constraints and unit processing times is strongly NP-hard [162]. Note that there is no benefit in preemption if $p_j = 1$ for all jobs $j \in N$. This implies that $P\,|\,pmtn,\,prec\,|\,C_{\max}$ is strongly NP-hard in general.

Via a reduction from Clique, Lenstra and Rinnooy Kan [112] proved that it is NP-hard to approximate $P \,|\, prec,\, p_j = 1 \,|\, C_{\max}$ better than $\frac{4}{3}$. In Section 2.3.2, we provide the same lower bound for $P \,|\, or\text{-}prec \,|\, C_{\max}$ using a reduction from Vertex Cover.

The problem $P2 \,|\, prec,\, p_j = 1 \,|\, C_{\max}$ is solvable in polynomial time [47, 32], but the complexity of $Pm \,|\, prec,\, p_j = 1 \,|\, C_{\max}$ for $m \geq 3$ remains open [53, 151, 12]. The best-known approximation factors for $P \,|\, prec,\, p_j = 1 \,|\, C_{\max}$ are $\frac{4}{3}$ $(m = 3)$ [32, 107, 22] and $2 - \frac{7}{3m+1}$ $(m \geq 4)$ [50]. The preemptive variant $P \,|\, pmtn,\, prec \,|\, C_{\max}$ can be approximated within $2 - \frac{2}{m}$ [129, 107]. Assuming a variant of the Unique Games Conjecture [100] together with a result of Bansal and Khot [15], Svensson [160] proved that an approximation factor of 2 is essentially best possible for $P \,|\, prec,\, p_j = 1 \,|\, C_{\max}$.

Levey and Rothvoss [118] and Garg [56] presented $(1 + \varepsilon)$-approximations for $Pm \,|\, prec,\, p_j = 1 \,|\, C_{\max}$ that run in time $n^{\mathcal{O}(r)}$ with $r = (\log n)^{\mathcal{O}(\log \log n)}$ and $r = (\log n)^{\mathcal{O}(1)}$ for fixed $\varepsilon > 0$, respectively. This was improved by Kulkarni et al. [104], who presented a $(1 + \varepsilon)$-approximation of similar running time as [118] for a variant of $Pm \,|\, pmtn,\, prec \,|\, C_{\max}$ where jobs may be preempted, but have to be continued on the *same machine*. The algorithms in [118, 56, 104] use *LP hierarchies*. Roughly speaking, these algorithms are LP based and the LP relaxation is strengthened in several rounds. The number of rounds equals the parameter r above.

If we restrict the precedence constraints to be an outforest, then the preemptive variant of AND-scheduling is solvable in polynomial time. These precedence graphs are of special interest to us. Recall that, in an outforest, every node has at most one predecessor. If the precedence graph is an outtree, requiring *all* predecessors to be completed and requiring *at least one* of them is equivalent. Thus, AND- and OR-precedence constraints coincide on outtrees.

A number of polynomial-time algorithms were proposed for preemptive AND-scheduling to minimize the makespan when the precedence constraints are in form of an outtree. Hu [86] proposed the first such algorithm for unit processing time jobs, and Brucker, Garey and Johnson [23] presented an algorithm that can also deal with non-trivial release dates. Monma [126] improved the running time of the algorithm in [23] from $\mathcal{O}(n \log n)$ to $\mathcal{O}(n)$. The first polynomial-time algorithm for $P \,|\, pmtn,\, prec = outtree \,|\, C_{\max}$ is due to Muntz and Coffman [130]. Later, Gonzalez and Johnson [65] proposed an algorithm that has a better asymptotic running time and uses fewer preemptions than the one in [130]. Finally, Lawler [109] presented a polynomial-time algorithm for the preemptive variant with release dates, i.e., for $P \,|\, r_j,\, pmtn,\, prec = outtree \,|\, C_{\max}$, which is an extension of the algorithm in [23].

Most of these algorithms solve a symmetric scheduling problem on intrees. The algorithm for outtrees is obtained by first solving the symmetric problem on the reverse partial order and then flipping the schedule. We elaborate on this symmetry and these algorithms in the next section. Johannes [93] presented a polynomial-time algorithm for $P \,|\, or\text{-}prec,\, p_j = 1 \,|\, C_{\max}$ that uses similar ideas as Hu [86]. In Section 2.4, we use the above algorithms to design polynomial-time algorithms for $P \,|\, r_j,\, pmtn,\, or\text{-}prec \,|\, C_{\max}$ and $P \,|\, r_j,\, or\text{-}prec,\, p_j = 1 \,|\, C_{\max}$.

2.2 Preliminaries

Note that $P \,|\, or\text{-}prec \,|\, C_{\max}$ is a generalization of $P \,||\, C_{\max}$, which is already strongly NP-hard [52]. If the precedence graph G is an outforest, then OR- and AND-precedence constraints on G are equivalent. So, the NP-hardness result of Du, Leung and Young [40] for chains implies that OR-scheduling to minimize the makespan remains strongly NP-hard, even if the number of machines is fixed.

Proposition 2.2 (Du-Leung-Young [40])
$Pm \,|\, or\text{-}prec = chains \,|\, C_{\max}$ is strongly NP-hard for all $m \geq 2$.

2.2.1 Related Algorithms for AND-Scheduling

List Scheduling. Graham's List Scheduling algorithm [66] not only applies for minimizing the makespan, but is used as a framework for many other scheduling problems. In principle, any scheduling algorithm that first chooses a linear ordering $\sigma : N \to [n]$ on the jobs and processes the jobs according to this order is a list scheduling algorithm. Note that List Scheduling never introduces unnecessary idle time. That is, if a machine is idle, then all unscheduled jobs are currently processing or there is no available job at that time. A schedule that is returned by List Scheduling is called a *list schedule*.

The general idea of showing that List Scheduling is a 2-approximation is to partition the time interval from 0 to $C_{\max}$ into two types of intervals: In the first type, all machines are busy and, in the second one, some machine is idle. Then, the length of each of these two types of intervals is bounded by the optimum makespan. We briefly show why List Scheduling is 2-approximate for $P \,|\, prec \,|\, C_{\max}$ in order to highlight similarities and differences in the analysis to our OR-scheduling model later.

Proposition 2.3 (Graham [66])
List Scheduling is a $\left(2 - \frac{1}{m}\right)$-approximation for $P \,|\, prec \,|\, C_{\max}$.

Proof. There are two obvious lower bounds on the optimal makespan, $C^*_{\max}$. First, no schedule can process more than m jobs at a time, so $C^*_{\max} \geq \frac{1}{m}\, p(N)$. Second, every job requires all its predecessors to be completed before it can start. Hence, the optimal makespan is at least as large as the total processing time along any path in the precedence graph $G = (N, E_\wedge)$. More precisely, for $j \in N$ and a path $S \subseteq N$ in G that ends at j, it holds that $C^*_{\max} \geq p(S)$.

Consider the schedule that is returned by List Scheduling, and let $k \in N$ be a job with $C_k = C_{\max}$. Let j_1 be a predecessor of k that completes last, i.e., $(j_1, k) \in E_\wedge$ and $C_{j_1} = \max\{C_j \,|\, (j,k) \in E_\wedge\}$. In the same way, we construct a sequence of predecessors, where j_q is chosen such that $(j_q, j_{q-1}) \in E_\wedge$ and $C_{j_q} = \max\{C_j \,|\, (j, j_{q-1}) \in E_\wedge\}$ for all $q \geq 1$, until we reach a job j_ℓ without predecessors. This sequence of jobs corresponds to a path $S = \{j_\ell, \ldots, j_1, k\} \subseteq N$ in G that ends at k.

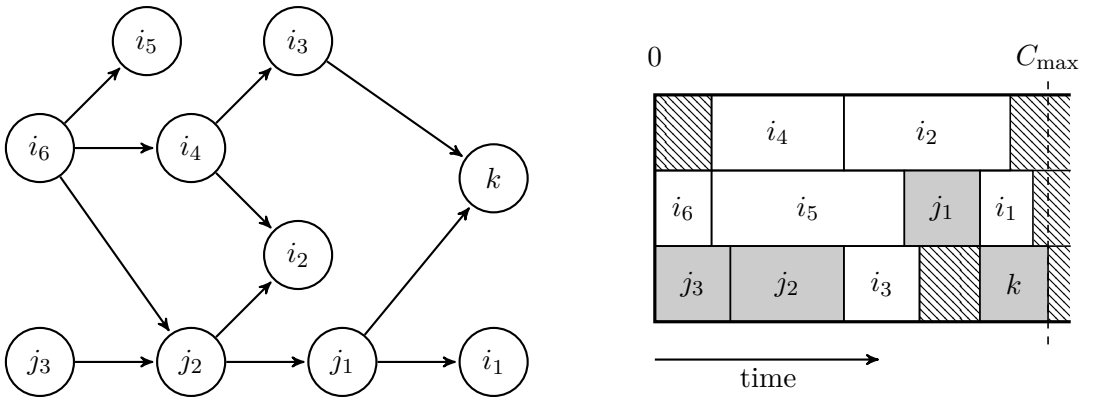

Fig. 2.1: An instance of $P\,|\,prec\,|\,C_{\max}$ with its precedence graph (left) and a feasible list schedule on three machines (right). The jobs in $S = \{j_3, j_2, j_1, k\}$ are highlighted in gray in the schedule.

Let I_S be the union of all intervals where a job in S is processed. Note that no two jobs in S run simultaneously as every job in S requires its predecessor in S to be completed before it can start. That is, the total length of these intervals is $|I_S| = p(S)$. Figure 2.1 illustrates a list schedule and such a sequence of jobs.

Let $I_B = [0; C_{\max}] \setminus I_S$ be the points in time when no job in S is being processed. We claim that in I_B, all machines are busy with jobs in $N \setminus S$. Suppose there is an idle interval $I = [s; t] \subseteq I_B$. Let $j_q \in S$ be the first job in S that starts after I, i.e., $S_{j_q} = \min\{S_j \,|\, j \in S,\, S_j \geq t\}$. By definition of $I \subseteq I_B$ and our choice of j_q, we know that the predecessor $j_{q+1} \in S$ of j_q completes before I, i.e., $C_{j_{q+1}} \leq s$. Since List Scheduling never introduces unnecessary idle time, there is a predecessor i of j_q that completes somewhere in the interval $[t; S_{j_q}]$, as otherwise we could have started j_q during I. This, however, contradicts the definition of S because then $C_{j_{q+1}} \leq s < t \leq C_i$, so job i would be part of S instead of j_{q+1}. Thus, all machines are busy during the intervals in I_B, and $|I_B| \leq \frac{1}{m}\, p(N \setminus S)$. In total, we obtain

$$\begin{aligned} C_{\max} = C_k &= |I_B| + |I_S| \leq \frac{1}{m} p(N \setminus S) + p(S) \\ &= \frac{1}{m} p(N) + \left(1 - \frac{1}{m}\right) p(S) \leq \left(2 - \frac{1}{m}\right) C^*_{\max}. \end{aligned} \tag{2.1}$$

This proves the claim. □

Similarly, one can prove that List Scheduling is a $(2 - \frac{1}{m})$-approximation for $P\,|\,r_j, prec\,|\,C_{\max}$, see [72]. The proof idea to show that List Scheduling is also 2-approximate for OR-scheduling is similar to the proof of Proposition 2.3. However, the analysis is slightly different, because we only know that *at least one* predecessor completes before a job.

The performance guarantee of List Scheduling in Proposition 2.3 is tight. Consider an instance with $2m-1$ jobs and no precedence constraints, i.e., $E_\wedge = \emptyset$. The processing times are $p_1 = \cdots = p_{m-1} = m-1$, $p_m = m$ and $p_{m+1} = \cdots = p_{2m-1} = 1$. An optimal solution is to pair jobs q and $m+q$ to run on one machine for all $q \in [m-1]$ and to schedule job m on its own machine. This gives a schedule of makespan equal to m. List Scheduling could first assign jobs $1, \ldots, m-1$ to the first $m-1$ machines and all jobs $m+1, \ldots, 2m-1$ to the last machine. At time $m-1$, all jobs except for the m-th job are completed. Then, job m is assigned to any machine, which yields a makespan of $2m-1$.

A Symmetric Problem for $P \,|\, prec \,|\, C_{\max}$. As already mentioned in Section 2.1, some algorithms for minimizing the makespan on outtrees [86, 130, 23, 109], actually consider a slightly different problem. In general, minimizing the makespan, $C_{\max} = \max\{C_j \,|\, j \in N\}$, can be seen as a special case of minimizing the *maximum lateness*, which is denoted by $\gamma = L_{\max}$ in the three-field notation of [68]. In this setting, each job $j \in N$ has a deadline $\overline{d}_j$ and its *lateness* is defined as $L_j \coloneqq \max\{0, C_j - \overline{d}_j\}$. The maximum lateness of a schedule is then $L_{\max} \coloneqq \max\{L_j \,|\, j \in N\}$. By letting $\overline{d}_j = 0$ for every $j \in N$, it becomes clear that minimizing the makespan is a special case of minimizing the maximum lateness.

There is yet another connection between the two objective functions, $C_{\max}$ and $L_{\max}$, when non-trivial release dates are involved. Consider the decision variants of $P \,|\, r_j, prec \,|\, C_{\max}$ and $P \,|\, prec \,|\, L_{\max}$.[10] For makespan minimization, we are given a time horizon $T \geq 0$ and the question is whether there exists a feasible schedule with makespan less or equal than T or not. The decision version of maximum lateness asks whether or not there is a feasible schedule with $L_{\max} = 0$.

Suppose we have a YES-instance of $P \,|\, r_j, prec \,|\, C_{\max}$, and consider a feasible schedule that obeys the precedence constraints, where every job starts after its release date and that has makespan less or equal than T. Now, we reverse the precedence constraints, i.e., if $(i,j) \in E_\wedge$ in the precedence graph, we introduce an arc (j,i) instead, and set the deadline of job j to $\overline{d}_j = T - r_j$. Note that the constructed instance is an instance of $P \,|\, prec \,|\, L_{\max}$. Further, if we reverse the feasible schedule of $P \,|\, r_j, prec \,|\, C_{\max}$, i.e., we set the starting time of job j to $T - C_j$ and its completion time to $T - S_j$, we obtain a feasible schedule for the corresponding instance of $P \,|\, prec \,|\, L_{\max}$ with $L_{\max} = 0$. Similarly, any YES-instance of $P \,|\, prec \,|\, L_{\max}$ can be transformed to a YES-instance of $P \,|\, r_j, prec \,|\, C_{\max}$ with $T = \max\{\overline{d}_j \,|\, j \in N\}$ and release dates $r_j = T - \overline{d}_j$.

The algorithms of Brucker, Garey and Johnson [23] and Lawler [109] actually solve $P \,|\, prec = intree, p_j = 1 \,|\, L_{\max}$ and $P \,|\, pmtn, prec = intree \,|\, L_{\max}$, respectively. With the above observations, we can use these algorithms to obtain algorithms for $P \,|\, r_j, prec = outtree, p_j = 1 \,|\, C_{\max}$ and $P \,|\, r_j, pmtn, prec = outtree \,|\, C_{\max}$, respectively. Given an AND-scheduling instance with release dates where the precedence

[10] Usually arbitrary deadlines are not explicitly listed in the three-field notation of $P \,|\, prec \,|\, L_{\max}$.

constraints are in the form of an outtree, we construct the corresponding flipped instance of minimizing the maximum lateness (with suitable T). The reversed precedence constraints are then in form of an intree, so we can apply the respective algorithms in [23, 109] to obtain an optimal solution, and reverse the resulting schedule.

Hu's algorithm [86] for $P\,|\,prec = outtree,\ p_j = 1\,|\,C_{\max}$ uses a similar idea. The setting is slightly easier, since all release dates are trivial, i.e., the flipped maximum lateness problem is actually an instance of $P\,|\,prec = intree,\ p_j = 1\,|\,C_{\max}$. The algorithm assigns a label to each job which equals the distance (number of jobs on the path) from that particular job to the unique root. In each step, a job with highest label among all unscheduled jobs is assigned to the next idle slot on a machine. Muntz and Coffman [130] combine this idea with *processor sharing*, i.e., a machine can process more than one job at a time, and use [123] to obtain a feasible preemptive schedule.

The algorithm of Brucker, Garey and Johnson [23] for $P\,|\,prec = intree,\ p_j = 1\,|\,L_{\max}$ first modifies the deadlines so that they are consistent with the intree-precedence constraints. That is, it starts with the root i, and updates the deadline of each predecessor j to $d'_j = \min\{\overline{d}_j, \overline{d}_i - 1\}$. This is then successively done for all jobs. The modified deadlines d'_j now have the property that $(i,j) \in E_\wedge$ implies $d'_i < d'_j$. If the original deadlines are trivial ($\overline{d}_j = d$ for all $j \in N$), then $d - d'_j$ are precisely the labels of Hu's algorithm [86]. Finally, the jobs are scheduled *earliest deadline first*, a rule that goes back to Jackson [91], with respect to the modified deadlines. Thereby, the schedule obeys the precedence constraints.

2.2.2 Earliest Start Schedules and Minimal Chains

To analyze the performance of our algorithms in Sections 2.3.1 and 2.4, we use the concept of what we call *minimal chains*. Informally, a minimal chain of a job k is a set of jobs that need to be scheduled so that k can complete as early as possible. To define these minimal chains properly, we use the notion of an *earliest start schedule*, see, e.g., [43, 125, 93]. Although these schedules are well-defined for general AND/OR-scheduling, we only need and define them in the OR-scheduling context.

Definition 2.4 (Earliest Start Schedule)
Let N be a set of jobs and $G = (N, E_\vee)$ a precedence graph. An *earliest start schedule* is defined as a schedule on an infinite number of machines such that

(i) a job j without predecessors starts at time r_j, and
(ii) a job j with $\mathcal{P}(j) \neq \emptyset$ starts at time $\max\{r_j, \min\{C_i \,|\, i \in \mathcal{P}(j)\}\}$.

Clearly, an earliest start schedule respects the OR-precedence constraints of the instance, i.e., it is feasible according to Definition 1.13. Also, the completion time of a job in any feasible schedule on m machines is bounded from below by its completion time in an earliest start schedule. That is, if C_j denotes the completion time of job j in an earliest start schedule, then the optimum makespan satisfies $C^*_{\max} \geq \max\{C_j \,|\, j \in N\}$.

Note that an earliest start schedule is not necessarily unique, but the start and completion times of all jobs are unique. Earliest start schedules can be constructed in polynomial time by iteratively scheduling every job as early as possible [43].

Definition 2.5 (Minimal Chains)
Let N be a set of jobs and $G = (N, E_\vee)$ a precedence graph. Let $k \in N$ and let C_j be the completion time of $j \in N$ in an earliest start schedule (Definition 2.4). A set $L \subseteq N$ is called a *minimal chain of* k if $L \in \mathcal{FS}$ is an inclusion-minimal feasible starting set with $k \in L$ and $\max_{j \in L} C_j = C_k$. The set of minimal chains of k is denoted by $\mathcal{MC}(k)$, and the *length of the minimal chain of* k is $mc(k) \coloneqq C_k$.

We can construct a minimal chain of k by iteratively tracing back predecessors that delay job k in an earliest start schedule. That is, starting at k, we mark one of its predecessors j with $C_j = S_k$, and then proceed with j in the same manner, i.e., we mark a predecessor i of j with $C_i = S_j$, and so on, until we reach a job i' that starts at its release date. If i' has no predecessors, we are done. If $\mathcal{P}(i') \neq \emptyset$, we mark a predecessor j' of i' with $C_{j'} \leq S_{i'}$, and continue with j' as described above. The marked jobs now correspond to a minimal chain of k. That is, a minimal chain $L = \{j_1, \dots, j_\ell\} \in \mathcal{MC}(k)$ is a path in G with $\mathcal{P}(j_1) = \emptyset$, $j_q \in \mathcal{P}(j_{q+1})$ for all $q \in [\ell - 1]$ and $j_\ell = k$ such that $S_{j_1} = r_{j_1}$ and $S_{j_q} = \max\{r_{j_q}, C_{j_{q-1}}\}$ for all $2 \leq q \leq \ell$. We call j_q *the predecessor of* j_{q+1} *in* L for $q \in [\ell - 1]$ and denote this by defining the set $\mathcal{P}_L(j_{q+1}) \coloneqq \{j_q\}$. A job $j_h \in L$ is said to *dominate* the minimal chain L if $mc(k) = r_{j_h} + \sum_{q=h}^{\ell} p_{j_q}$.

Example 2.6 (Minimal Chains)
Let $N = \{j_1, j_2, j_3, j_4, j_5, j_6, j_7, j_8, k\}$ with processing times $p_{j_1} = p_{j_6} = p_k = 1$, $p_{j_2} = p_{j_3} = p_{j_5} = p_{j_7} = 2$, $p_{j_4} = 3$, $p_{j_8} = 4$ and release dates $r_{j_1} = 2$, $r_{j_2} = 1$, $r_{j_6} = 4$, $r_j = 0$ for all other jobs j. The precedence graph G and an earliest start schedule are depicted in Figure 2.2.

The set of minimal chains of k is $\mathcal{MC}(k) = \{\{j_2, j_6, j_7, k\}, \{j_3, j_5, j_6, j_7, k\}\}$ with $mc(k) = 8$. The chain $\{j_2, j_6, j_7, k\}$ is dominated by j_6, and $\{j_3, j_5, j_6, j_7, k\}$ is dominated by jobs j_3 and j_6. Note that $\{j_1, j_4, j_7, k\} \in \mathcal{FS}$ with $p(\{j_1, j_4, j_7, k\}) = 7$, but if we would want to schedule k via this chain, the processing is delayed by the release date of job j_1. The predecessor of k in both minimal chains is $\mathcal{P}_{L_1}(k) = \mathcal{P}_{L_2}(k) = \{j_7\}$.

If the minimal chains are constructed as described above, the jobs preceding a job j in a minimal chain $L \in \mathcal{MC}(k)$ form a minimal chain of j. This motivates the following definition.

Definition 2.7 (Closed Collection of Minimal Chains)
Let N be a set of jobs, $G = (N, E_\vee)$ a precedence graph and $L_k \in \mathcal{MC}(k)$ for all $k \in N$. The collection of minimal chains $\{L_k \mid k \in N\}$ is called *closed* if $j \in L_k$ implies $L_j \subseteq L_k$.

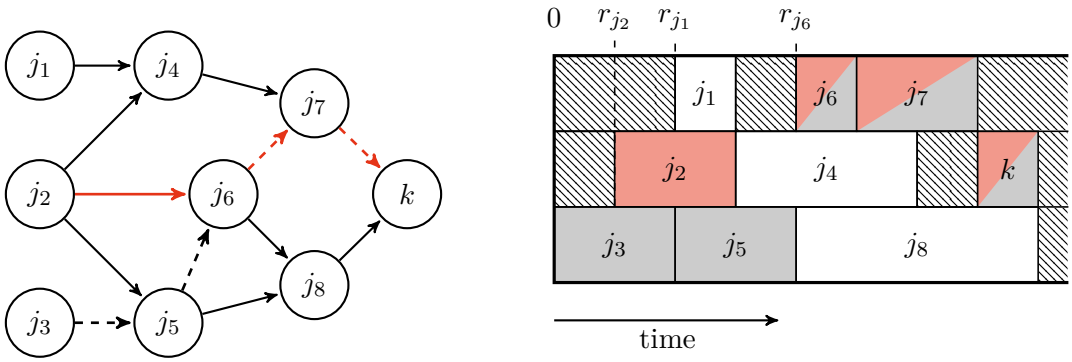

Fig. 2.2: The instance in Example 2.6 with precedence graph G (left) and an earliest start schedule (right). The paths in G that correspond to the minimal chains in $\mathcal{MC}(k)$ are depicted dashed and red, respectively. Jobs in minimal chains are highlighted in gray and red, respectively. Dashed red arcs and two-colored jobs are contained in both minimal chains.

2.3 Approximability and Hardness of the Non-Preemptive Variant

We first show that Graham's List Scheduling algorithm [66] is a 2-approximation algorithm for $P \,|\, r_j,\, or\text{-}prec \,|\, C_{\max}$ in Section 2.3.1, and then show that already the problem without release dates is APX-hard in Section 2.3.2.

2.3.1 List Scheduling is a 2-Approximation Algorithm

In the following, we denote the optimal makespan by $C^*_{\max}$. There are two obvious lower bounds on the optimal makespan. First, any feasible schedule cannot do better than splitting the total processing load equally among all machines, so $C^*_{\max} \geq \frac{1}{m}\, p(N)$. Second, every job requires at least one of its predecessors to be completed before it can start. If we start with an empty schedule, the earliest possible completion time of job j is, by definition, equal to its completion time in the earliest start schedule. Thus, $C^*_{\max} \geq \max\{mc(j) \,|\, j \in N\}$.

Erlebach, Kääb and Möhring [43] presented a 2-approximation algorithm for AND/OR-scheduling without release dates, $P \,|\, ao\text{-}prec \,|\, C_{\max}$. The algorithm first computes the earliest start schedule, i.e., each job without predecessors starts at time 0, and each job j with predecessors starts at time $\max\{\min\{C_i \,|\, i \in \mathcal{P}(j)\}, \max\{C_i \,|\, (i,j) \in E_\wedge\}\}$. Then, for every job j with $\mathcal{P}(j) \neq \emptyset$, an OR-predecessor with $C_i = S_j$ is fixed to be an AND-predecessor, i.e., the arc $(i,j) \in E_\wedge$ is introduced. Finally, List Scheduling is applied to the AND-instance $P \,|\, prec \,|\, C_{\max}$ with precedence graph $G = (N, E_\wedge)$.

We show that also without transforming the instance, List Scheduling is 2-approximate for OR-precedence constraints, even with non-trivial release dates. The proof uses ideas of [72] and the proof of Proposition 2.3.

Theorem 2.8
List Scheduling is a $\left(2 - \frac{1}{m}\right)$*-approximation for* $P \,|\, r_j,\ or\text{-}prec \,|\, C_{\max}$.

Proof. Consider the schedule that is returned by List Scheduling, and let $k \in N$ be a job with $C_k = C_{\max}$. If there is no idle time in the interval $[0; S_k]$, then a similar chain of inequalities as (2.1) with $S = \{k\}$ yields $C_{\max} \leq \left(2 - \frac{1}{m}\right) C^*_{\max}$.

So suppose there is idle time, and let I be the union of all intervals in which some machine is idle. Let $S \subseteq N$ be a set of jobs such that $\{j^{\text{in}}\} \cup S$ is a path in G^{in} from the initial job to k. At every point in time $t \in I$, a job in S is either not yet released, or is currently running on some machine. Otherwise, there is an unscheduled available job in S that can be processed at time t.

Let $L' \in \mathcal{MC}(k)$ be a minimal chain of k and index the jobs $L' = \{j_1, \dots, j_\ell\}$ such that $j_\ell = k$, $\mathcal{P}(j_1) = \emptyset$, and $j_q \in \mathcal{P}(j_{q+1})$ for all $q \in [\ell - 1]$. Recall that L' is a path in G. So, at every idle point in time, some job of L' is either being processed or not yet released. That is, the total idle time is $|I| \leq mc(k)$. Now, let $h \in [\ell]$ be maximal such that j_h dominates the minimal chain L', i.e., $mc(k) = r_{j_h} + \sum_{q=h}^{\ell} p_{j_q}$. Let $L = \{j_h, \dots, j_\ell\}$, and consider the points in time $I_L = [0; r_{j_h}] \cup \bigcup_{j \in L} [S_j; C_j]$ when j_h is not yet released or some job in L is being processed.[11]

W.l.o.g., we can assume that at least one machine is running during $[0; r_{j_h}]$. Otherwise, if all machines were idle at some point $t \in [0; r_{j_h}]$, then all jobs j with $r_j \leq t$ are already completed at time t. Thus, also in the optimum solution, no machine is running at time t. So, we can disregard these time slots where no machine is running at all. That is, during $I_B = [0; C_{\max}] \setminus I_L$, all machines are busy with jobs in $N \setminus L$ and the total processing load of jobs that are running in I_B is less or equal than $p(N \setminus L) - r_{j_h}$. Hence, $|I_B| \leq \frac{1}{m}\left(p(N \setminus L) - r_{j_h}\right)$ and we obtain

$$\begin{aligned} C_{\max} = C_k = |I_B| + |I_L| &\leq \frac{1}{m}\left(p(N \setminus L) - r_{j_h}\right) + r_{j_h} + \sum_{q=h}^{\ell} p_{j_q} \\ &= \frac{1}{m}\, p(N) + \left(1 - \frac{1}{m}\right) mc(k) \leq \left(2 - \frac{1}{m}\right) C^*_{\max} \end{aligned} \tag{2.2}$$

This proves the claim. □

Corollary 2.9
List Scheduling solves $1 \,|\, r_j,\ or\text{-}prec \,|\, C_{\max}$ *optimally.*

[11] In contrast to the proof of Proposition 2.3, the intervals $[S_j; C_j]$ for $j \in L$ are not necessarily disjoint, because jobs in L might run in parallel. That is, instead of $|I_S| = p(S)$, as in the proof of Proposition 2.3, we get $|I_L| \leq r_{j_h} + p(L)$.

2.3.2 An Inapproximability Result

In the remainder of this section, we derive a lower bound on the approximability of OR-scheduling without release dates under the assumption P ≠ NP. To obtain this lower bound, we reduce the NP-complete Vertex Cover problem to $P\,|\,or\text{-}prec\,|\,C_{\max}$. We show that if we could approximate $P\,|\,or\text{-}prec\,|\,C_{\max}$ within some factor $\eta < \frac{4}{3}$, we could solve Vertex Cover in polynomial time.

To define Vertex Cover, we need the notion of an *undirected graph*, see [36]. An *undirected graph* $\mathcal{G} = (V, \mathcal{E})$ consists of a finite set of *vertices* V and a set of *edges* $\mathcal{E} \subseteq \{\{v, u\}\,|\,v, u \in V,\, v \neq u\}$, which are subsets of cardinality two of V.[12] If $\{v, u\} \in \mathcal{E}$ is an edge, we say that $u, v \in V$ are *incident* to $\{v, u\}$.

Definition 2.10 (Vertex Cover)
Let $\mathcal{G} = (V, \mathcal{E})$ be an undirected graph. A *vertex cover* is a subset $W \subseteq V$ such that $\{v, u\} \cap W \neq \emptyset$ for all $\{v, u\} \in \mathcal{E}$.

For $K \in \mathbb{N}$ and an undirected graph $\mathcal{G}$, the Vertex Cover problem asks whether there is a vertex cover of size at most K. Vertex Cover is NP-complete [98], even when restricted to, e.g., undirected graphs with maximum degree equal to 3 [54]. Dinur and Safra [37] proved that it is NP-hard to approximate Vertex Cover within a factor of 1.36. Gavril (see [53], page 134), Bar-Yehuda and Even [21] and Hochbaum [82] presented simple 2-approximation algorithms, and under a variant of the Unique Games Conjecture [100], a factor of 2 is essentially best possible [101]. Our main result in this section is that OR-scheduling has the same inapproximability bound as AND-scheduling, see [112].

Theorem 2.11
It is NP-hard to $\left(\frac{4}{3} - \varepsilon\right)$*-approximate* $P\,|\,or\text{-}prec\,|\,C_{\max}$ *for any fixed* $\varepsilon > 0$.

To prove Theorem 2.11, we first describe a reduction from any Vertex Cover instance to an instance of $P\,|\,or\text{-}prec\,|\,C_{\max}$. Then, we show that Vertex Cover is a Yes-instance if and only if the corresponding instance of OR-scheduling has an optimum makespan of 3 (Lemma 2.13). Hence, if we had an η-approximation algorithm for $P\,|\,or\text{-}prec\,|\,C_{\max}$ with $\eta < \frac{4}{3}$, we could use this algorithm to obtain a feasible schedule with makespan at most $\eta \cdot 3 < 4$. Since all input data in the instance are integer, the makespan of any feasible schedule is also integer. So, a feasible solution with makespan strictly less than 4 actually has makespan ≤ 3, i.e., the solution is optimal. Thus, we could use the η-approximation algorithm to obtain an optimum solution and decide whether the initial Vertex Cover instance is a Yes-instance, implying P = NP.

[12] We distinguish between directed graphs $G = (N, E)$, as in Section 1.1.2, and undirected graphs $\mathcal{G} = (V, \mathcal{E})$ via the nomenclature of *nodes* versus *vertices* and *arcs* versus *edges*, respectively. Note that arcs are tupels $(i, j) \in E \subseteq N \times N$ and edges are sets $\{v, u\} \in \mathcal{E} \subseteq 2^V$.

We can assume $K < |V|$ and $|\mathcal{E}| > 0$, as otherwise Vertex Cover is trivial. Further, if $K \geq |\mathcal{E}|$, then we can choose an incident vertex for every edge to be in the vertex cover. So, w.l.o.g., we can assume $K < \min\{|V|, |\mathcal{E}|\}$.

The Reduction. Consider an instance of Vertex Cover, i.e., let $\mathcal{G} = (V, \mathcal{E})$ be an undirected graph and $K \in [\min\{|V|, |\mathcal{E}|\} - 1]$. We now describe the construction of the instance of $P \,|\, or\text{-}prec \,|\, C_{\max}$. The number of machines is $m = |\mathcal{E}| + |V| - K$, and the set of jobs consists of four disjoint sets, $N = J_K \,\dot\cup\, J_V \,\dot\cup\, J_{\mathcal{E}} \,\dot\cup\, X$. We refer to Example 2.12 for an instance of Vertex Cover and the corresponding OR-scheduling instance.

In J_K, we introduce K jobs of unit processing time. The sets J_V and $J_{\mathcal{E}}$ contain a job j_v and $j_{\{v,u\}}$ for every vertex $v \in V$ and every edge $\{v, u\} \in \mathcal{E}$, respectively. The processing times of all jobs $j_v \in J_V$ and $j_{\{v,u\}} \in J_{\mathcal{E}}$ are equal to 1. We set the predecessors of an edge-job in $J_{\mathcal{E}}$ to be the jobs corresponding to its incident vertices, i.e., $\mathcal{P}(j_{\{v,u\}}) = \{j_v, j_u\}$ for all $\{v, u\} \in \mathcal{E}$. Moreover, we assign OR-precedence constraints in $J_K \times J_V$ in the form of an outforest such that each job in J_V is successor of exactly one job in J_K. This can be done since $|J_K| = K < |V| = |J_V|$. It is not important which job in J_K is the predecessor of which job in J_V. We only need these precedence constraints to ensure that no job in J_V can start at time 0 and all jobs in J_V are available as soon as all jobs in J_K are completed. The remaining jobs in X are dummy jobs to enforce a certain structure of any optimal schedule. The set X contains $m - K$ jobs of processing time equal to 2, and these jobs do not have any predecessors or successors. Note that $X \neq \emptyset$ because $K < \min\{|V|, |\mathcal{E}|\}$.

Example 2.12 (Reduction to $P \,|\, or\text{-}prec \,|\, C_{\max}$)
Let $\mathcal{G} = (\{v_1, v_2, v_3, v_4\}, \{\{v_1, v_3\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_3, v_4\}\})$ and $K = 2$. Figure 2.3 illustrates the precedence graph of the corresponding OR-scheduling instance and a feasible schedule. It holds $|V| = |\mathcal{E}| = 4$, so $m = 6$, $|X| = 4$ and $|N| = 14$. Note that the jobs $\{j_{v_2}, j_{v_3}\}$ scheduled in the interval $[1; 2]$ correspond to a vertex cover of size 2.

In fact, the instance of $P \,|\, or\text{-}prec \,|\, C_{\max}$ is constructed such that any feasible schedule of makespan ≤ 3 has a similar structure. If the makespan is ≤ 3, then all jobs in $J_{\mathcal{E}}$ need to be scheduled in the interval $[2; 3]$. Further, all jobs in J_K and X have to start at time 0. The cardinality of the set X ensures that there are exactly K slots in $[1; 2]$ left for jobs in J_V. Since all jobs in $J_{\mathcal{E}}$ have to be available at time 2, the vertex-jobs in J_V that are placed in $[1; 2]$ correspond to a vertex cover, see Lemma 2.13. The remaining vertex-jobs can be scheduled in $[2; 3]$ in parallel to the edge-jobs in $J_{\mathcal{E}}$.

The following lemma together with the discussion after Theorem 2.11 completes the proof of Theorem 2.11.

Lemma 2.13
Let $\mathcal{G} = (V, \mathcal{E})$ be an undirected graph and $K \in [\min\{|V|, |\mathcal{E}|\} - 1]$. Vertex Cover is a Yes-instance if and only if the corresponding instance of $P \,|\, or\text{-}prec \,|\, C_{\max}$ has makespan ≤ 3.

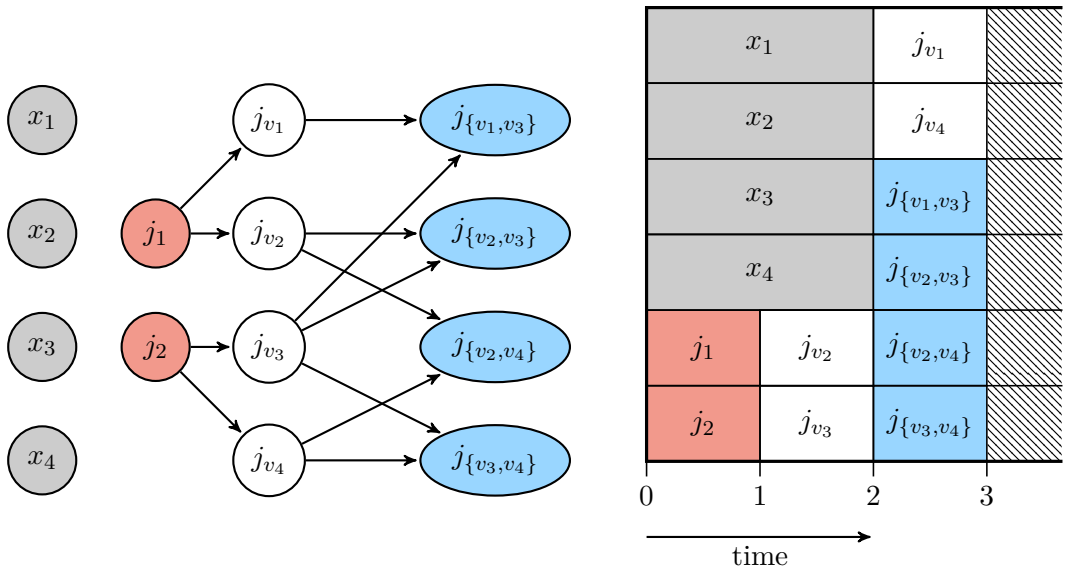

Fig. 2.3: The precedence graph of the instance of $P\,|\,or\text{-}prec\,|\,C_{\max}$ in the reduction from Vertex Cover for the instance in Example 2.12 (left) and a feasible schedule with makespan ≤ 3 (right). To highlight the structure of the schedule, jobs in J_K, $J_{\mathcal{E}}$ and X are depicted red, blue and gray, respectively.

Proof. Suppose Vertex Cover is a Yes-instance. Let $W \subseteq V$ be a vertex cover of size $|W| \leq K$, and let $J_W \subseteq J_V$ be the jobs corresponding to W. We can schedule the jobs in a similar way as in Figure 2.3 (right). That is, we schedule all jobs in X and J_K in the intervals $[0;2]$ and $[0;1]$, respectively. So all jobs in J_V are available at time 1. In $[1;2]$, there are exactly $m - |X| = K$ slots left in which we can schedule the jobs in J_W and some other jobs of $J_V \setminus J_W$ if $|W| < K$. Hence, at time 2, all jobs in $J_{\mathcal{E}}$ are available and can be scheduled in $[2;3]$. Finally, we schedule the remaining jobs in J_V on the remaining $m - |\mathcal{E}| = |V| - K$ machines in the interval $[2;3]$.

Now suppose that the instance of $P\,|\,or\text{-}prec\,|\,C_{\max}$ has makespan ≤ 3. Recall that $m = |\mathcal{E}| + |V| - K$ and note that the total processing load of all jobs is

$$\sum_{j \in N} p_j = |J_K| + |J_V| + |J_{\mathcal{E}}| + 2|X| = K + |V| + |\mathcal{E}| + 2\,(m - K) = 3m. \tag{2.3}$$

So any feasible schedule with makespan ≤ 3 has a makespan of exactly 3 and there is no idle time within the interval $[0;3]$. Consider an optimal schedule of makespan equal to 3. Due to the precedence constraints, no job in J_V can start before time 1, and no job in $J_{\mathcal{E}}$ can start before time 2. So, the jobs $J_{\mathcal{E}}$ are scheduled in the interval $[2;3]$. Also, the jobs in J_V that are scheduled in $[1;2]$ correspond to a vertex cover, since otherwise not all jobs in $J_{\mathcal{E}}$ would be available at time 2. It remains to be shown that the set of jobs of J_V scheduled in $[1;2]$ has cardinality at most K.

The total processing load of all jobs is equal to $3m$, see (2.3). As there is no idle time within $[0;3]$, exactly m jobs start at time 0. The only jobs that can start at time 0 are those in $J_K \cup X$ (all other jobs have predecessors). Since $|J_K| + |X| = K + m - K = m$, we know that the jobs in J_K and X are scheduled in $[0;1]$ and $[0;2]$, respectively. This leaves exactly $m - |X| = K$ slots in $[1;2]$ in which only jobs of J_V are scheduled. □

Technically, it would suffice to let J_K be a singleton that precedes all jobs in J_V. However, choosing $|J_K| = K$, and, thus, the total processing load to be equal to $3m$ makes the argument slightly easier. Note that the processing times in the OR-scheduling instance in the reduction are in $\{1, 2\}$ only. This is indeed necessary, i.e., we cannot get a reduction for unit processing times. Also, the proof does not work if we allow preemption. In this case, we could preempt a job in X at time 1, and process more than K jobs of J_V in the interval $[1;2]$. In fact, as we see in the next section, we can solve the variants with unit processing times or preemption in polynomial time.

2.4 A Polynomial-Time Algorithm for the Preemptive Variant

In contrast to the non-preemptive setting, an optimal preemptive schedule never has idle time if there are available jobs. Without preemption, it could make sense to leave a machine idle, although there is an available job j, if, e.g., a long job i becomes available soon. If preempting jobs is allowed, we could just schedule a fraction of j, and once i becomes available, we preempt j and process i.

The main result of this section is a polynomial-time algorithm for the preemptive variant of minimizing the makespan, $P \,|\, r_j,\, pmtn,\, or\text{-}prec \,|\, C_{\max}$. Recall that all processing times and release dates of jobs are non-negative integers. So, preemptive and non-preemptive scheduling of unit processing time jobs is equivalent, since there is no need to preempt. Thus, a polynomial-time algorithm for the preemptive variant also solves $P \,|\, r_j,\, or\text{-}prec,\, p_j = 1 \,|\, C_{\max}$.

The algorithm is summarized in Algorithm 1 and works as follows. First, we compute a closed collection of minimal chains $\{L_j \,|\, j \in N\}$ with $L_j \in \mathcal{MC}(j)$ for all $j \in N$ (Definition 2.7). Then, we define a directed graph G' by removing all arcs from the precedence graph that are not part of the paths that correspond to the minimal chains. Observe that G' is an outforest if the collection of minimal chains is closed. Now, we apply Lawler's algorithm [109] for the AND-instance $P \,|\, r_j,\, pmtn,\, prec = outtree \,|\, C_{\max}$ with precedence graph G' to compute a preemptive schedule. The returned schedule is feasible for the initial instance, because G' is a subgraph of the original precedence graph. For unit processing times, we could use the algorithms of Brucker, Garey and Johnson [23] or Monma [126] instead of Lawler's algorithm [109].

The idea of using an earliest start schedule, constructing an outforest G' and solving the related AND-instance on G' was previously used in [93] to get a polynomial-time algorithm for $P \,|\, or\text{-}prec,\, p_j = 1 \,|\, C_{\max}$. Algorithm 1 extends this idea and uses the notion of minimal chains to handle also non-trivial release dates and preemption.

Input: Instance of $P \,|\, r_j,\ pmtn,\ or\text{-}prec \,|\, C_{\max}$
Output: A feasible schedule for $P \,|\, r_j,\ pmtn,\ or\text{-}prec \,|\, C_{\max}$

1 Construct an earliest start schedule;
2 Compute a closed collection $\{L_j \,|\, j \in N\}$ with $L_j \in \mathcal{MC}(j)$ for all $j \in N$;
3 $E' \leftarrow \emptyset$;
4 **for** $j \in N$ **do**
5 Enumerate the jobs in $L_j = \{j_1, \ldots, j_\ell\}$ so that $j_q \in \mathcal{P}_{L_j}(j_{q+1})$ for all $q \in [\ell - 1]$;
6 $E' \leftarrow E' \cup \{(j_1, j_2), (j_2, j_3), \ldots, (j_{\ell-1}, j_\ell)\}$;
7 **end**
8 Set $G' = (N, E')$;
9 Apply Lawler's algorithm [109] for $P \,|\, r_j,\ pmtn,\ prec = outtree \,|\, C_{\max}$ on G';
10 **return** schedule returned by Lawler's algorithm [109];

Algorithm 1: An algorithm for $P \,|\, r_j,\ pmtn,\ or\text{-}prec \,|\, C_{\max}$.

Observe that the algorithm in [93] cannot be extended to the preemptive case. Although one could split all jobs into a sequence of unit processing time jobs, the size of the resulting instance is exponential in the size of the initial instance. Instead of showing optimality of the schedule returned by Algorithm 1 directly, we analyze the structure of an optimal preemptive schedule. More precisely, we show that, for any closed collection of minimal chains, there is an optimal preemptive schedule that is feasible for the transformed precedence graph G'. Correctness of Algorithm 1 then follows from the correctness of Lawler's algorithm [109], see Theorem 2.18.

To analyze the structure of an optimal schedule, we need some additional notation. If jobs are allowed to preempt, we want to "keep track" of how much of the minimal chain of a job is already processed at every point in time. To formalize this, we consider an equivalent instance of unit processing time jobs by splitting every job. Note that the size of this new instance is not polynomial in the size of the initial instance. However, this is not a problem because we never explicitly construct the new instance. It is used only for the analysis of Algorithm 1.

Definition 2.14 (Preemptive Instance)
Let N be a set of jobs and $G = (N, E_\vee)$ a precedence graph. The *preemptive instance* on the job set $N^{(p)}$ is defined as follows:

(*i*) for every $j \in N$ introduce p_j jobs $j_1, \ldots, j_{p_j} \in N^{(p)}$ of unit processing time,
(*ii*) for all $j \in N$, set $\mathcal{P}(j_1) = \{i_{p_i} \,|\, (i, j) \in E_\vee\}$ and $\mathcal{P}(j_q) = \{j_{q-1}\}$ for all $2 \leq q \leq p_j$, and
(*iii*) set $r_{j_q} = r_j$ for all $q \in [p_j]$ and $j \in N$.

W.l.o.g., we neglect jobs of zero processing time, as they can be scheduled as soon as they become available without increasing the load on that machine. Note that, if all jobs have unit processing time, then $N^{(p)} = N$. Every feasible (non-preemptive) schedule for the preemptive instance corresponds to a feasible preemptive schedule for the initial instance, and vice versa. We informally extend the definitions of feasible starting sets (Definition 1.14) and minimal chains (Definition 2.5) to *fractions of jobs* via the original definitions on the preemptive instance. Note that (the lengths of) all minimal chains coincide with the non-preemptive instance. That is, if $i \in L_j \in \mathcal{MC}(j)$ for $j \in N$, then, for $j_q \in N^{(p)}$ with $q \in [p_j]$, there is $L_{j_q} \in \mathcal{MC}(j_q)$ such that $i_1, \dots, i_{p_i} \in L_{j_q}$. We call these minimal chains $L_{j_1} \in \mathcal{MC}(j_1), \dots, L_{j_{p_j}} \in \mathcal{MC}(j_{p_j})$ the *natural extensions* of $L_j \in \mathcal{MC}(j)$. The collection of the natural extensions $\{L_j \mid j \in N^{(p)}\}$ is closed if and only if the corresponding collection $\{L_j \mid j \in N\}$ is closed.

Definition 2.15 (Inverted Jobs)
Let N be a set of jobs and $G = (N, E_\vee)$ a precedence graph. Let $j \in N$, $L \in \mathcal{MC}(j)$ and let $i \in L$ be the predecessor of j in L, i.e., $\mathcal{P}_L(j) = \{i\}$. Consider a feasible schedule with completion times C_k for every $k \in N$. We call the pair (i, j) *inverted* if $C_i \geq C_j$. The number of inverted pairs in the schedule is denoted by I_C.

Our approach for proving correctness of Algorithm 1 is as follows. Suppose we have an optimal preemptive schedule that is not feasible for the AND-instance on the constructed precedence graph G'. Then the corresponding schedule of the preemptive instance contains at least one inverted pair. Lemma 2.16 describes a procedure that swaps two jobs $i, k \in N^{(p)}$ that are scheduled consecutively. We apply this procedure to successively swap two jobs until the inverted pair is not inverted anymore. The procedure does not cause additional inversions and does not increase the makespan, so we obtain an optimal schedule with strictly fewer inverted pairs. Thereby, we show that there always exists an optimal solution without inversions (Lemma 2.17).

For the notation of Lemma 2.16, we neglect release dates. We describe how to incorporate release dates in the proof of Lemma 2.17, which together with Lemma 2.16 proves correctness of Algorithm 1. Recall that all jobs in $N^{(p)}$ have unit processing time, i.e., $S_j = C_j - 1$ for all $j \in N^{(p)}$ and any schedule.

Lemma 2.16
Let $\{L_j \mid j \in N^{(p)}\}$ be a closed collection of minimal chains and let C^*_j be the completion time of $j \in N^{(p)}$ in a feasible schedule for the preemptive instance. Let $i \in N^{(p)}$ with $C^*_i \geq 2$, and let $J_i^- = \{j \in N^{(p)} \mid C^*_j = C^*_i - 1\}$ be the jobs scheduled directly before i. Assume that $|J_i^-| = m$ and $C^*_j \leq C^*_i - 2$ for $j \in \mathcal{P}_{L_i}(i)$.[13] Then, there is $k \in J_i^-$ such that swapping i and k, i.e., setting $C'_i = C^*_i - 1 = C^*_k$, $C'_k = C^*_k + 1 = C^*_i$ and $C'_j = C^*_j$ for all $j \in N^{(p)} \setminus \{i, k\}$, yields a feasible schedule with $C'_{\max} = C^*_{\max}$ and $I_{C'} \leq I_{C^*}$.

[13] So, moving i to $[C^*_i - 2; C^*_i - 1]$ does not violate its precedence constraints or cause an inversion.

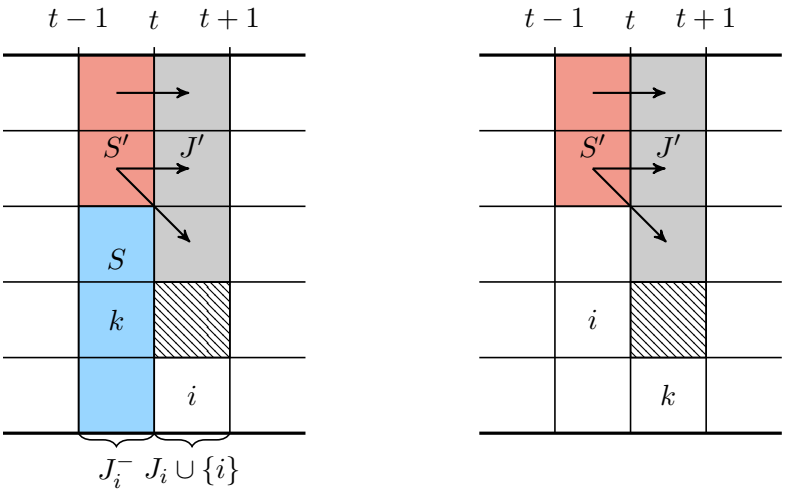

Fig. 2.4: Relevant time slots $[t-1; t+1]$ in the initial schedule (left) and final schedule (right), respectively. The arcs indicate that the respective job in S' is the predecessor of the corresponding job in J'. The sets S', J' and S are depicted in red, gray and blue, respectively. In this example, $J = \emptyset$, so $J_i = J'$.

Proof. To shorten notation, set $t = C_i^* - 1$. Note that the makespan does not change if we swap two unit processing time jobs. Let $J_i = \{j \in N^{(p)} \setminus \{i\} \mid C_j^* = t+1\}$ be the jobs running in parallel to i on the other machines. Note that $|J_i| \leq m-1$, and recall that there are $|J_i^-| = m$ jobs that are being processed directly before i. For $j \in N^{(p)}$, let $\mathcal{A}_j = \{j' \in N^{(p)} \mid C_{j'}^* < C_j^*\}$ be the set of jobs that complete before j starts.

Let $J' = \{j \in J_i \mid \mathcal{P}_{L_j}(j) \cap J_i^- \neq \emptyset\} \cup \{j \in J_i \mid |\mathcal{P}(j) \cap \mathcal{A}_j| = 1 \text{ and } \mathcal{P}(j) \cap \mathcal{A}_j \subseteq J_i^-\}$ be the set of jobs that are scheduled parallel to i and that are processed directly after their predecessor in the minimal chain or directly after the only predecessor preceding them in the schedule. Let $S' \subseteq J_i^-$ be the set of these predecessors of jobs in J'. We do not want to swap i with a job in S', since this would cause an inversion or yield an infeasible schedule. Note that $|S'| \leq |J'|$, and let $S = J_i^- \setminus S'$ and $J = J_i \setminus J'$. Then, $|S| = m - |S'| \geq m - |J'| \geq |J_i| + 1 - |J'| = |J| + 1 \geq 1$, so $S \neq \emptyset$. We claim that any $k \in S$ satisfies the statement. Figure 2.4 illustrates the sets and the corresponding schedules before and after swapping i and k.

Let $k \in S$, and consider the resulting schedule after swapping k and i. Feasibility of the initial schedule implies that at most m jobs are running at any point in time. Scheduling k one time slot later does not violate any precedence constraints or cause an additional inversion by definition of S. Also, moving i one time slot forward is feasible and does not cause an inversion by assumption. All other jobs remain unchanged, so the resulting schedule is feasible and $I_{C'} \leq I_{C^*}$. □

For the following lemma, we consider the instance with the initial job j^{in}, which is the only job without predecessors, i.e., the set of jobs is $N^{(p)} \cup \{j^{\text{in}}\}$ and $p_{j^{\text{in}}} = 0$.

Lemma 2.17
Let $\{L_j \mid j \in N^{(p)}\}$ be a closed collection of minimal chains $L_j \in \mathcal{MC}(j)$ for all $j \in N^{(p)}$. There exists an optimal schedule of the preemptive instance with no inverted pairs.

Proof. Recall that all processing times of jobs in $N^{(p)}$ are equal to 1. Consider an optimal schedule with completion times C^*_j for all $j \in N^{(p)}$ such that the number of inversions I_{C^*} is minimal among all optimal solutions, and suppose that $I_{C^*} \geq 1$. We show how to construct a schedule with $C'_{\max} = C^*_{\max}$ and $I_{C'} < I_{C^*}$ using Lemma 2.16.

Since the schedule is optimal, the initial job j^{in} starts at time 0. Let $j \in N^{(p)}$ and $i \in \mathcal{P}_{L_j}(j)$ such that (i, j) is an inverted pair, i.e., $C^*_i \geq C^*_j \geq mc(j) \geq mc(i) + 1$. We index the jobs in $L_j = \{j_0, j_1, \ldots, j_\ell, j_{\ell+1}\} \in \mathcal{MC}(j)$ such that $j^{\text{in}} = j_0$, $i = j_\ell$, $j = j_{\ell+1}$ and $j_{q-1} \in \mathcal{P}_{L_j}(j_q)$ for all $q \in [\ell + 1]$. Note that $mc(j_{q-1}) + 1 \leq mc(j_q) \leq C^*_{j_q}$ for all $q \in [\ell + 1]$ and $mc(j_0) = mc(j^{\text{in}}) = 0$.

Using Lemma 2.16, we move the jobs $j_1, \ldots, j_\ell$ successively (in this order) to the front such that they complete at times $mc(j_1), \ldots, mc(j_\ell)$, respectively. For all $k \in N^{(p)}$, it holds

$$C^*_k \geq mc(k) \geq r_k + p_k = r_k + 1. \tag{2.4}$$

So, we can swap those jobs $k \in \{j_1, \ldots, j_\ell\}$ that do not complete at time $mc(k)$ to the front without violating their respective release date. Thereby, we obtain a schedule that satisfies

$$0 = C'_{j_0} < mc(j_1) = C'_{j_1} < mc(j_2) = C'_{j_2} < \cdots < mc(j_\ell) = C'_{j_\ell} < C'_j. \tag{2.5}$$

Since we first move job j_1 to the front, then j_2, and so on, we ensure that, when we apply Lemma 2.16 for $i = j_q$ (in the notation of Lemma 2.16), then its predecessor j_{q-1} completes at time $mc(j_{q-1}) < mc(j_q)$. So, the assumptions of Lemma 2.16 are satisfied. The procedure of Lemma 2.16 does not violate any release dates, since $k \in S$ (in the notation of Lemma 2.16) is scheduled later and it is feasible to schedule j_q earlier due to (2.4) for all $q \in [\ell]$.

Figure 2.5 illustrates the current completion times and the time slots in which we move the jobs in the minimal chain L_j. Note that it is not necessary to move the job $j = j_{\ell+1}$. However, by applying Lemma 2.16, it might happen that $k = j$ (in the notation of Lemma 2.16) is chosen, i.e., j is "passively moved back". Similarly, a job j_h might be "passively moved back" when we swap j_q with $q < h$ to the front. This is not a problem, since we deal with j_h in a later iteration.

Multiple application of Lemma 2.16 ensures that the resulting schedule is feasible and has no more inversions than the initial schedule. Further, Lemma 2.16 implies $C'_{\max} = C^*_{\max}$, and $I_{C'} < I_{C^*}$ because i and j are not inverted anymore, see (2.5). This contradicts the choice of the initial schedule being an optimal solution with fewest inversions. Hence, there exists an optimal solution without inversions, which proves the claim. □

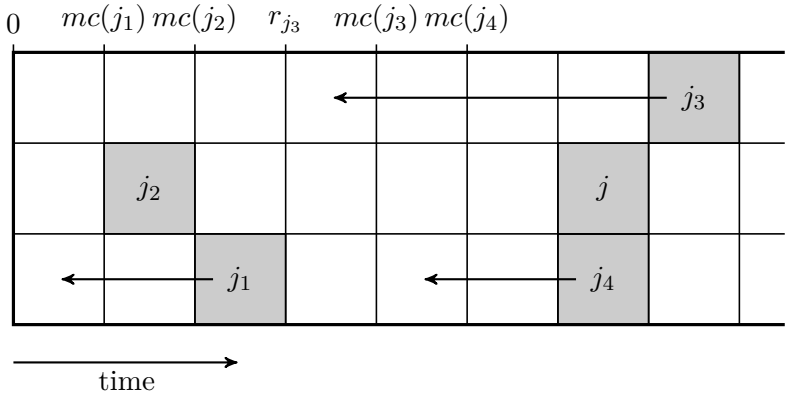

Fig. 2.5: Illustration of the procedure to move jobs in $L_j \setminus \{j\} = \{j_1, j_2, j_3, j_4\}$ to the front. Blank squares are jobs not in L_j. Arrows indicate into which time slot we want to move the respective jobs. The jobs are moved "lowest index first" rather than all at once. Note that $mc(j_3) > mc(j_2) + 1$ because $r_{j_3} = 3$.

We are now ready to prove our main theorem in this section.

Theorem 2.18
Algorithm 1 solves $P \,|\, r_j$, pmtn, or-prec$\,|\, C_{\max}$ to optimality in polynomial time.

Proof. First, observe that the digraph G' constructed by Algorithm 1 is a subgraph of the initial precedence graph G. Since the schedule returned by the algorithm is feasible for the AND-instance on G' (this follows from correctness of Lawler's algorithm [109]), it is feasible for the OR-instance on G. Construction of the earliest start schedule and Lawler's algorithm run in polynomial time [43, 109]. Also, we can compute the closed collection of minimal chains and construct G' in polynomial time. So, Algorithm 1 runs in polynomial time and returns a feasible schedule.

As for optimality of the schedule returned by Algorithm 1, let $\{L_j \,|\, j \in N\}$ be the closed collection of minimal chains that is computed in line 2, and let G' be the corresponding subgraph of G. Since $\{L_j \,|\, j \in N\}$ is closed, G' is an outforest. Thus, OR- and AND-precedence constraints on G' are equivalent.

Consider the schedule returned by Algorithm 1, i.e., by Lawler's algorithm [109] on G', and let $C_{\max}$ be its makespan. Since the schedule is feasible for the OR-instance with precedence graph G', it is also feasible for the initial precedence graph G. By Lemma 2.17, there exists an optimal solution with makespan $C^*_{\max}$ for the instance on G that is also feasible for the instance on G'. Since the schedule returned by Algorithm 1 is optimal for the instance on G', it holds $C_{\max} \leq C^*_{\max}$. This proves the claim. □

Recall that, for unit processing time jobs, there is no need to preempt. In this case, we could replace Lawler's algorithm [109] in Algorithm 1 by, e.g., one of the algorithms in [23, 126]. The following corollary extends the result of [93] to release dates.

Corollary 2.19
Algorithm 1 solves $P \,|\, r_j$, *or-prec*, $p_j = 1 \,|\, C_{\max}$ *to optimality in polynomial time.*

2.5 Open Problems

Since the preemptive problem can be solved in polynomial time (Theorem 2.18), it seems natural to focus on the non-preemptive variant. Note that the approximation factor $2 - \frac{1}{m}$ of List Scheduling (Theorem 2.8) is strictly less than 2, but approaches 2 as m increases. An obvious research question is to get the approximation factor down to $2 - \varepsilon$ for some fix $\varepsilon > 0$. Conversely, one could also try to find stronger lower bounds than the one presented in Theorem 2.11.

Problem 2.20
Close the gap of $\left[\frac{4}{3}; 2\right]$ in the approximation guarantee for $P \,|\, or\text{-}prec \,|\, C_{\max}$.

The performance guarantees in Theorems 2.8 and 2.18 match the best-known ones for minimizing the makespan if the precedence graph is an outtree. Recall that, in this special case, AND-scheduling and OR-scheduling coincide. Clearly, any improvement for OR-scheduling directly transfers to AND-scheduling on outtrees. On the other hand, due to its close connection with (closed collections of) minimal chains, any progress on the approximation factor for AND-scheduling on outtrees might also be applicable to OR-scheduling.

Problem 2.21
Is there a $(2 - \varepsilon)$-approximation for $P \,|\, prec = outtree \,|\, C_{\max}$ for a fix $\varepsilon > 0$?

Recall that AND-scheduling and OR-scheduling to minimize the makespan remain strongly NP-hard, even when the number of machines is fixed [40]. Nonetheless, there are $(1 + \varepsilon)$-approximation algorithms with slightly super-polynomial running time for AND-scheduling of unit processing time jobs [118, 56] and a variant of preemption [104]. OR-scheduling of unit processing time jobs or with preemption can be solved in polynomial time, even when the number of machines is not fixed, see Theorem 2.18. However, it is not clear whether better approximation ratios than 2 are possible for non-preemptive OR-scheduling if the number of machines is fixed.

Problem 2.22
Is there a $(2 - \varepsilon)$-approximation for $Pm \,|\, or\text{-}prec \,|\, C_{\max}$ for a fix $\varepsilon > 0$?

Note that the inapproximability result of Theorem 2.11 does not apply if m is constant. Fixing the number of machines in the reduction would mean that we fix the number of vertices in the undirected graph. In this case, we can solve Vertex Cover by full enumeration of all $|2^V| = 2^{|V|}$ (which is a constant) candidates for vertex covers. Recall that, unless P = NP, an FPTAS for OR-scheduling is out of reach, even if we fix the number of machines [52, 40]. Nevertheless, there could be η-approximation algorithms with $\eta < \frac{4}{3}$ or even $\eta = 1 + \varepsilon$ for some arbitrary small but fix $\varepsilon > 0$.

Problem 2.23
Is there a PTAS for $Pm \,|\, or\text{-}prec \,|\, C_{\max}$?

Svensson [160] discovered an interesting connection between makespan minimization and minimizing the sum of weighted completion times for AND-scheduling. He showed that, under a variant of the Unique Games Conjecture [100], if there is no $(2 - \varepsilon')$-approximation algorithm for minimizing the sum of weighted completion times on a single machine, then there is no $(2 - \varepsilon)$-approximation algorithm for makespan minimization. Finding a similar connection for OR-scheduling, or between AND-scheduling and OR-scheduling, might solve some of the aforementioned open problems.

Chapter 3

Combinatorial Algorithms for the Sum of Weighted Completion Times

In this chapter, we present combinatorial exact and approximate algorithms for variants of min-sum set cover and OR-scheduling on a single machine to minimize the sum of weighted completion times. Parts of this chapter coincide with work that is available online in [75, 77]. The results in Section 3.4 are work in progress [76].

3.1 Related Work and Our Results

The problems considered in this chapter are special cases of generalized min-sum set cover (Definition 1.10) and variants of $1 \,|\, \textit{or-prec} \,|\, \sum w_j C_j$. Recall the min-sum covering problems of Section 1.2.1 and their connection to OR-scheduling, see Figure 1.5.

In Section 3.2, we introduce further variants of (generalized) min-sum set cover, and we revisit some important definitions and results from AND-scheduling. We continue with analyzing the frontier between problems that are solvable in polynomial time and NP-hard problems. More precisely, we show that bipartite OR-scheduling is strongly NP-hard, even when restricted to unit processing times or unit weights (Section 3.3), and present polynomial-time algorithms for variants of generalized min-sum set cover if the family of sets $\mathcal{R}$ is *laminar* (Section 3.4.1). We also propose a 2-approximation algorithm for GMSSC on laminar sets in Section 3.4.2. In Section 3.5.1, we provide a general framework to obtain approximation algorithms for scheduling problems using a *density-maximizing* Greedy algorithm. Finally, we propose two applications of this framework in the context of OR-scheduling and present 4-approximation algorithms for $1 \,|\, \textit{or-prec} = \textit{bipartite} \,|\, \sum w_j C_j$ and $1 \,|\, \textit{or-prec} = \textit{intree} \,|\, \sum w_j C_j$ in Section 3.5.2. We begin with reviewing some related work. Figure 3.2 in Section 3.2.1 gives an overview of the problems that are discussed in this chapter and the related problems in the literature, which we describe briefly in the following paragraphs.

Min-Sum Set Cover and Related Problems. The basic problem is min-sum set cover (Definition 1.9), which was first studied in [44]. Min-sum set cover (MSSC) is related to the *chromatic sum* problem that was introduced by Kubicka and Schwenk [103].

Here, we are given an undirected graph $\mathcal{G} = (V, \mathcal{E})$, and the task is to find a feasible coloring, i.e., a partition of the vertices $V = V_1 \dot\cup \cdots \dot\cup V_\ell$ such that no two adjacent vertices are assigned the same color, i.e., for all $\{v, u\} \in \mathcal{E}$, $v \in V_q$ implies $u \notin V_q$. The objective is to find a feasible coloring that minimizes $\sum_{q=1}^{\ell} q \cdot |V_q|$.

Note that chromatic sum can be seen as a special case of MSSC, see also [44, 45]: For an undirected graph $\mathcal{G} = (V, \mathcal{E})$, we introduce a set in $\mathcal{R}$ for every vertex in V and an element in U for every *independent set* of $\mathcal{G}$.[14] The set corresponding to $v \in V$ contains all elements that correspond to independent sets containing v. However, this reduction is not of polynomial size, as one would have to list all independent sets of $\mathcal{G}$. Observe that min-sum set cover and chromatic sum are min-sum variants of the well-known Set Cover (or the equivalent Hitting Set) problem and the Chromatic Number problem, respectively. These problems were shown to be NP-complete by Karp [98], and it is NP-hard to obtain constant-factor approximations for Set Cover [39] and Chromatic Number [169].

Chromatic sum can be solved in linear time for trees [103]. For bipartite graphs, the problem is APX-hard and admits a $\frac{10}{9}$-approximation [20].[15] Bar-Noy et al. [18] showed that greedily picking an independent set of maximum cardinality and removing those vertices gives a 4-approximation for arbitrary graphs, assuming that one can find such an independent set in polynomial time. If a maximum independent set can be approximated within a factor of $\eta \geq 1$, the algorithm in [18] yields a 4η-approximation for chromatic sum. Bar-Noy, Halldórsson and Kortsarz [19] provided an example for which the approximation ratio of 4 of the Greedy algorithm is tight.

Feige, Lovász and Tetali [44] observed that applying the Greedy algorithm of [18] to min-sum set cover, which is to choose the element in U that is contained in most uncovered sets next, yields a 4-approximation for MSSC. They simplified the proof by analyzing the performance ratio via a time-indexed linear program (see Section 4.2.3) instead of comparing the Greedy solution directly to the optimum. In the journal version [45] of their paper, they further simplified the proof to an elegant histogram framework, which inspired the result in Section 3.5, and they proved that one cannot approximate min-sum set cover strictly better than 4, unless P = NP.

The histogram proof of [45] was generalized by Iwata, Tetali and Tripathi [90] to obtain a 4-approximation for the *minimum linear ordering* problem, which generalizes min-sum set cover. Other generalizations of MSSC and the results in [45] were proposed in, e.g., [97, 159]. More recently, a similar proof was used in [122] to get a $4\sqrt{|U|}$-approximation for precedence-constrained min-sum set cover (Definition 1.11), and in [81] to get an 8-approximation algorithm for *expanding search*, a problem that was introduced by Alpern and Lidbetter [6]. Happach, Hellerstein and Lidbetter [75] generalized the histogram proof of [90] to linear ordering problems with *subadditive* cost functions or *superadditive* weight functions and proposed new applications of this

[14] An *independent set* of an undirected graph $\mathcal{G}$ is a subset of vertices $S \subseteq V$ such that the induced subgraph $\mathcal{G}[S] \coloneqq (S, \mathcal{E} \cap 2^S)$ contains no edges, see [36].

[15] An undirected graph $\mathcal{G} = (A \dot\cup B, \mathcal{E})$ is *bipartite* if $\{v, u\} \in \mathcal{E}$ implies $v \in A$ and $u \in B$.

histogram framework.[16] In Section 3.5.1, we present a special case of the framework in [75] in the context of single-machine scheduling problems, which generalizes the histogram proofs in [45, 122, 81]. We then provide two applications for OR-scheduling and obtain 4-approximation algorithms for restricted precedence graphs in Section 3.5.2.

The special case of min-sum set cover where every set in $\mathcal{R}$ contains at most two elements is called *min-sum vertex cover* (MSVC) and is APX-hard [45]. The Greedy algorithm for MSSC is 4-approximate for MSVC as well, and this analysis is tight [45]. Feige, Lovász and Tetali [44, 45] provided a 2-approximation algorithm for MSVC that is based on a time-indexed linear program and uses randomized rounding. Bansal et al. [13] used the same LP to obtain a 16/9-approximation for MSVC.

Munagala et al. [128] generalized MSSC by introducing non-negative costs for each element in U and non-negative weights for each set in $\mathcal{R}$. This problem, which is called *pipelined set cover*, is formally defined in Section 3.2.1. Among other things, the authors in [128] proved that the natural extension of the Greedy algorithm of [45] for min-sum set cover still yields a 4-approximation for pipelined set cover.

Recall the generalized min-sum set cover or GMSSC (Definition 1.10) that was introduced in [10]. The extreme cases $\kappa(R) = 1$ and $\kappa(R) = |R|$ are min-sum set cover and the minimum latency set cover problem, respectively. The latter was studied in [79] and is closely related to single-machine scheduling with AND-precedence constraints [166]. Over time, several constant-factor approximations for generalized min-sum set cover were proposed. Bansal, Gupta and Krishnaswamy [14] presented the first such algorithm with an approximation guarantee of 485, which was improved to 28 by Skutella and Williamson [156] and then to 12.4 by Im, Sviridenko and Zwaan [88]. The algorithms in [14, 156] use the same time-indexed linear program and the algorithm in [88] is based on a so-called configuration LP. The currently best-known approximation ratio for generalized min-sum set cover is 4.642 and is due to Bansal et al. [13], who use the canonical time-indexed LP of [14]. In Section 3.4, we present polynomial-time and approximation algorithms for variants of GMSSC if the family of sets $\mathcal{R}$ is *laminar*.

Related Work in Scheduling. The first polynomial-time algorithm for scheduling jobs on a single machine to minimize the sum of weighted completion times is due to Smith [157]. With AND-precedence constraints, the problem is strongly NP-hard via a reduction from Linear Arrangement [108, 112]. It remains strongly NP-hard, even if we restrict to unit processing times [112] or bipartite precedence graphs [166]. Various 2-approximation algorithms based on linear programs [148, 71, 30] (see also Sections 4.1 and 4.2) as well as combinatorial algorithms [25, 120] exist for $1\,|\,prec\,|\,\sum w_j C_j$. Assuming a variant of the Unique Games Conjecture [100], Bansal and Khot [15] showed that an approximation ratio of 2 is essentially best possible.

[16] A set function $g : 2^N \to \mathbb{Z}$ is called *subadditive* if $g(S \cup R) \leq g(S) + g(R)$ for all $S, R \subseteq N$. A function $g : 2^N \to \mathbb{Z}$ is called *superadditive* if $g(S \cup R) \geq g(S) + g(R)$, i.e., if $-g$ is subadditive.

AND-scheduling, i.e., $1\,|\,prec|\,\sum w_jC_j$, is solvable in polynomial-time for certain precedence graphs. Horn [85] proposed the first polynomial-time algorithm for inforests and outforests, which was improved in running time by Adolphson and Hu [2]. Forests are special cases of so-called *series-parallel* graphs, see, e.g., [161]. If the precedence graph is series-parallel, then $1\,|\,prec|\,\sum w_jC_j$ can be solved in polynomial time [108]. The key idea in [108] is to use the natural decomposition of a series-parallel graph to concatenate and merge optimal schedules for smaller instances to obtain an optimal schedule for the initial instance. Similar techniques led to polynomial-time algorithms for AND-scheduling problems with series-parallel precedence graphs and other objective functions, see, e.g., [1, 127]. Optimality of Lawler's algorithm [108] was reproved in [63] using so-called *two-dimensional Gantt charts* of [42]. We elaborate on these two-dimensional Gantt charts in Section 3.2.2. In a series of papers [30, 34, 7], the result of Lawler [108] was generalized to two-dimensional partial orders, see Section 4.2.2.

Sidney [152] proved a fundamental structural property of optimal solutions for $1\,|\,prec|\,\sum w_jC_j$, which now is known as the *Sidney decomposition*. More precisely, he showed that there exists an optimal solution that subsequently schedules a *density-maximizing initial set* of the so far unscheduled jobs, see Section 3.2.2. Lawler [108] presented a polynomial-time algorithm to get such density-maximizing initial sets through minimum cut computations, see also [135]. However, this does not give a polynomial time algorithm, since Sidney's result [152] does not specify the order of the jobs within each density-maximizing set. The algorithm in [108] was used independently by Chekuri and Motwani [25] and Margot, Queyranne and Wang [120] to obtain a 2-approximation algorithm for $1\,|\,prec|\,\sum w_jC_j$ that successively computes a density-maximizing initial set and schedules the jobs within this set in any feasible order. In Section 3.5, we use a similar idea for OR-scheduling.

Scheduling on parallel machines with OR-precedence constraints and unit processing time jobs to minimize the sum of completion times, $P\,|\,or\text{-}prec,\,p_j = 1\,|\,\sum C_j$, can be solved in polynomial time [93]. However, once we want to minimize the sum of *weighted* completion times, the single-machine problem with unit processing times already becomes strongly NP-hard [93]. In Section 3.3, we extend this result by showing that the problem remains NP-hard, even if we restrict the weights to be 0/1.

3.2 Preliminaries

We introduce a generalization of MSSC and a special case of GMSSC with restricted covering requirements in Section 3.2.1. Then, in Section 3.2.2, we recall some related results for AND-scheduling that are based on *density-maximizing sets*. An important notion that we use in Section 3.4 are *laminar sets*.

Definition 3.1 (Laminar Sets)
A family of sets $\mathcal{R}$ is called *laminar* if $R \cap S \in \{\emptyset, R, S\}$ for all $R, S \in \mathcal{R}$.

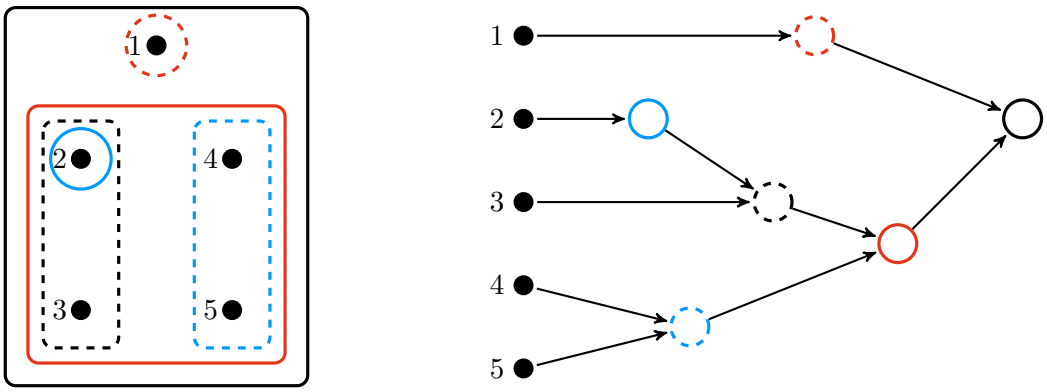

Fig. 3.1: A collection of laminar sets over $U = [5]$ (left) and the corresponding inforest-representation (right). The sets are depicted as colored dashed and solid lines. For simplicity, the nodes in the digraph corresponding to the sets are depicted in the same style as the respective sets on the left.

If $\mathcal{R}$ is laminar, then for every $R \in \mathcal{R}$ that is contained in some $R' \in \mathcal{R}$, i.e., $R \subsetneq R'$, there is a unique inclusion-minimal set $S \in \mathcal{R}$ with $S \supsetneq R$. This motivates the following *inforest-representation* of a collection of laminar sets $\mathcal{R} \subseteq 2^U$ over a finite set of elements U: We introduce a node for every element in U and every set in $\mathcal{R}$. The successor of $e \in U$ is the unique inclusion-minimal set in $\mathcal{R}$ that contains e. Similarly, the successor of a set in $\mathcal{R}$ is its (unique) inclusion-minimal superset in $\mathcal{R}$. Figure 3.1 depicts a laminar set family and the corresponding inforest-representation.

3.2.1 Pipelined Set Cover and All-But-Constant Min-Sum Set Cover

Consider an instance of min-sum set cover with a finite set of elements U and a collection of subsets $\mathcal{R} \subseteq 2^U$. Munagala et al. [128] proposed the following generalization of MSSC where every element $e \in U$ and every set $R \in \mathcal{R}$ is associated with a cost $c_e \in \mathbb{N}$ and weight $w_R \in \mathbb{N}$, respectively. For a given linear order $\pi : U \to [|U|]$, the *covering cost* of $R \in \mathcal{R}$ is defined as

$$\pi_c(R) := \min \left\{ \sum_{\substack{f \in U \\ \pi(f) \leq \pi(e)}} c_f \,\middle|\, e \in R \right\}. \tag{3.1}$$

Since the costs are positive, the covering cost of $\pi_c(R)$ is attained for the element in R that appears first in the linear ordering. In particular, for unit costs, the covering cost and covering time of a set coincide, i.e., $\pi_c(R) = \pi(R)$.

Definition 3.2 (Pipelined Set Cover)
Let U be a finite set of elements with costs $c_e \in \mathbb{N}$ for $e \in U$, and let $\mathcal{R} \subseteq 2^U$ with weights $w_R \in \mathbb{N}$ for $R \in \mathcal{R}$. The task is to find a linear ordering $\pi : U \to [|U|]$ that minimizes the sum of weighted covering costs, $\sum_{R \in \mathcal{R}} w_R \, \pi_c(R)$.

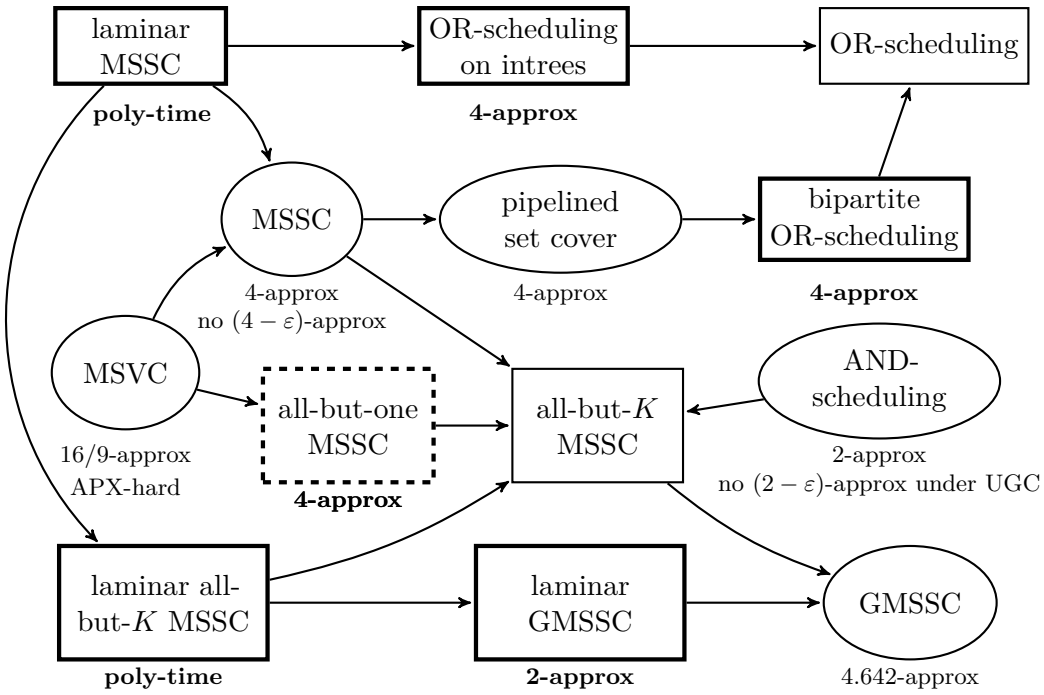

Fig. 3.2: Overview of related problems and results. An arrow from problem Π_1 to Π_2 indicates that Π_2 generalizes Π_1. Problems in rectangular frames are explicitly considered in this thesis, and our results are depicted in bold. All-but-one MSSC (rectangular dashed frame) is discussed in Chapter 4.

Note that min-sum set cover is the special case of pipelined set cover with unit costs and unit weights. We represent an instance of pipelined set cover via its covering graph $H = (U \cup \mathcal{R}, E)$ and interpret this as a bipartite OR-scheduling instance. Then, each job that corresponds to an element $e \in U$ has processing time c_e and zero weight, and each job that corresponds to a set $R \in \mathcal{R}$ has zero processing time and weight w_R. Natural questions that arise in the context of scheduling problems are whether approximation guarantees still hold for arbitrary processing times and how much the input data have to be restricted to obtain a problem that is solvable in polynomial time. In Section 3.5.2, we prove that the approximation factor of 4 for pipelined set cover can be preserved for bipartite OR-scheduling in general. Further, we observe that bipartite OR-scheduling remains strongly NP-hard, even if we restrict to unit processing times or unit weights in Section 3.3.

Recall the definition of generalized min-sum set cover from Definition 1.10. In Section 3.4, we show that if we restrict to a laminar family of sets $\mathcal{R} \subseteq 2^U$, the following special case of GMSSC becomes solvable in polynomial time.

Definition 3.3 (All-But-K Min-Sum Set Cover)
Let U be a finite set of elements, $K \in \mathbb{N}_0$ and $\mathcal{R} \subseteq 2^U$ with covering requirements $\kappa : \mathcal{R} \to \mathbb{N}$ where $\kappa(R) = \max\{1, |R| - K\}$ for all $R \in \mathcal{R}$. The task is to find a linear ordering $\pi : U \to [|U|]$ that minimizes the sum of covering times, $\sum_{R \in \mathcal{R}} \pi(R)$.

For this natural special case between min-sum set cover ($\kappa(R) = 1$) and minimum latency set cover ($\kappa(R) = |R|$), the covering time of every set is the first point in time when all but a constant of its elements have appeared in the linear ordering. If a set contains less than K elements, it is covered by its first element. In Section 4.3.2, we present an LP based 4-approximation for the special case where $K = 1$, i.e., for *all-but-one MSSC*. Note that MSSC is a special case of all-but-K MSSC if we choose K suitably large. Recall that, for MSVC, every set in $\mathcal{R}$ is of cardinality two, so MSVC is not only a special case of MSSC, but also of all-but-one MSSC. For $K = 0$, we obtain minimum latency set cover (MLSC) [79], which has the same approximability threshold as AND-scheduling [166].

If the collection of sets $\mathcal{R}$ is laminar, we highlight this by speaking of, e.g., *laminar min-sum set cover* instead of MSSC. When we represent an instance of laminar MSSC via its inforest-representation, it becomes clear that laminar MSSC is a special case of OR-scheduling on intrees to minimize the sum of weighted completion times (with the element-jobs having unit processing time and zero weight, the set-jobs having zero processing time and unit weight and an additional dummy job with zero processing time and weight that forms the root of the intree). Figure 3.2 illustrates the connection of the problems considered in this chapter, as well as the currently best-known upper and lower bounds on the approximation factors.

3.2.2 Density-Maximizing Initial Sets for AND-Scheduling

For a subset of jobs $S \subseteq N$, we define its *density* to be the ratio of its total weight and its total processing time, $\rho(S) := \frac{w(S)}{p(S)}$. Smith [157] showed that, for one of the most basic scheduling problems, $1 \,||\, \sum w_j C_j$, an optimal solution can be achieved by scheduling the jobs in non-increasing order of their density $\frac{w_j}{p_j}$. (The density $\frac{w_j}{p_j}$ is also known as the *Smith ratio* of $j \in N$.) That is, enumerate the jobs such that $\frac{w_1}{p_1} \geq \frac{w_2}{p_2} \geq \cdots \geq \frac{w_n}{p_n}$, and schedule them in the order $1 \to 2 \to \cdots \to n$. Today, this algorithm is known as *Smith's rule*. We include the proof of the following statement, as it has a neat geometric interpretation that we use later.

Proposition 3.4 (Smith [157])
Smith's rule (schedule the jobs in non-increasing order of $\frac{w_j}{p_j}$) is optimal for $1 \,||\, \sum w_j C_j$.

Proof. We prove the claim using an interchange argument and the two-dimensional Gantt charts of [42], see also [63, 114]. To this end, we introduce a rectangle of height w_j and width p_j for every job $j \in N$. Consider a feasible schedule that processes the jobs in order $j_1 \to j_2 \to \cdots \to j_n$ and let C_j be the completion time of job $j \in N$.

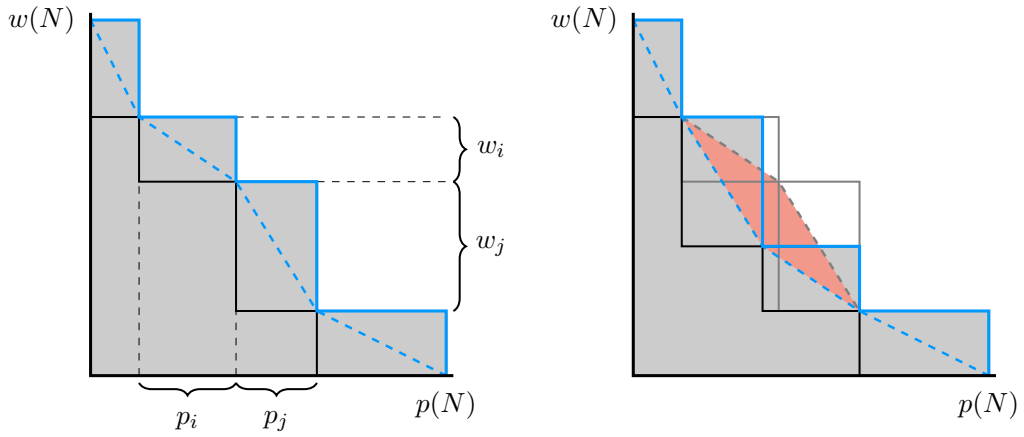

Fig. 3.3: Geometric interpretation of a schedule as a two-dimensional Gantt chart (left). The gray area underneath the upper envelope (blue solid line) equals the sum of weighted completion times. The slopes of the diagonals (blue dashed lines) are the negative Smith ratios. If we swap two consecutive jobs, we obtain the Gantt chart on the right. The area highlighted in red between the diagonals of the new (blue dashed) and old (gray dashed) schedule equals the difference in the objective values.

We place the rectangle of j_1 in a two-dimensional coordinate system so that its upper left corner is at $(0, w(N))$ and its lower right corner has coordinates $(p_{j_1}, w(N \setminus \{j_1\}))$. In a similar way, we align the rectangles of $j_2, j_3, \ldots, j_n$ in this order such that the upper left corner touches the lower right corner of the previous rectangle. That is, for $q \in \{2, \ldots, n\}$, the upper left and lower right corner of the rectangle of j_q are at $(C_{j_{q-1}}, w(\{j_q, \ldots, j_n\}))$ and $(C_{j_q}, w(\{j_{q+1}, \ldots, j_n\}))$, respectively. Note that the lower right corner of the last rectangle, which corresponds to j_n, has coordinates $(p(N), 0)$. Figure 3.3 (left) illustrates this arrangement. Observe that the slope of each rectangle, i.e., the diagonal from its upper left to its lower right corner, equals the negative Smith ratio of the corresponding job. The area underneath the upper envelope of the rectangles equals the objective value of the schedule, $\sum_{q=1}^{n} w_{j_q} C_{j_q}$.

Consider any optimal schedule that is not in line with Smith's rule, and let $i \to j$ be two consecutive jobs with $\frac{w_i}{p_i} < \frac{w_j}{p_j}$. If we swap the two rectangles corresponding to i and j, we obtain the picture on the right of Figure 3.3. Note that no rectangles other than those of i and j are altered. Further, the area between the upper envelope of the rectangles and their diagonals does not depend on the order of the rectangles. That is, only the area underneath the diagonals changes if we swap i and j. The difference in the objective values equals the negative of the area between the diagonals of the old and new schedule, which is highlighted in red in Figure 3.3 (right). This area is strictly positive, since $\frac{w_j}{p_j} > \frac{w_i}{p_i}$, which is a contradiction to the optimality of the initial schedule. Note that the objective value (the area of the Gantt chart) of any solution in line with Smith's rule is equal, which proves the claim. □

Note that, in a schedule that is in line with Smith's rule, the diagonals of the rectangles in the corresponding two-dimensional Gantt chart form a convex piecewise linear function, see Figure 3.3 (right). Observe that the slope of the diagonal from $(0, w(N))$ to $(p(N), 0)$ equals $-\rho(N) = -\frac{w(N)}{p(N)}$. Similarly, for any set of jobs $S \subseteq N$ that are scheduled consecutively, the slope of the diagonal from the upper left corner of the rectangle of the first job in S to the lower right corner of the rectangle of the last job in S equals the negative density of S.

Sidney [152] generalized Smith's rule to AND-scheduling. A set $S \subseteq N$ is called *initial* with respect to (w.r.t.) the precedence graph $G = (N, E_\wedge)$ if $j \in S$ and $(i, j) \in E_\wedge$ imply $i \in S$. Let $N = S_1 \dot\cup S_2 \dot\cup \cdots \dot\cup S_\ell$ be a partition such that, for all $h \in [\ell]$, $\bigcup_{q=1}^{h} S_q$ is initial w.r.t. G and $\rho(S_h) \geq \rho(S)$ for every $S \subseteq N \setminus \bigcup_{q=1}^{h-1} S_q$ that is initial w.r.t. $G[N \setminus \bigcup_{q=1}^{h-1} S_q]$. There exists an optimal schedule for $1\,|\,prec|\,\sum w_j C_j$ that schedules the jobs in the order $S_1 \to S_2 \to \cdots \to S_\ell$, where the jobs within each subset are scheduled in an optimal order [152]. The partition $S_1 \dot\cup S_2 \dot\cup \cdots \dot\cup S_\ell$ is called a *Sidney decomposition* of N. Chekuri and Motwani [25] and, independently, Margot, Queyranne and Wang [120] combined Sidney's result with the following observation.

Proposition 3.5 (Checkuri-Motwani [25], Margot-Queyranne-Wang [120]) *Suppose that N is a density-maximizing initial set for an instance of $1\,|\,prec|\,\sum w_j C_j$, i.e., $\rho(N) \geq \rho(S)$ for all initial $S \subseteq N$. Then any feasible schedule is a 2-approximate solution.*

Proof. The following proof was proposed in [63]. Consider an optimal solution and its two-dimensional Gantt chart, see, e.g., Figure 3.3. For $\ell \in [n]$, we denote the first ℓ jobs in the schedule by S_ℓ. Since the schedule is feasible, S_ℓ is initial and $\rho(S_\ell) \leq \rho(N)$ by assumption. Hence, in the two-dimensional Gantt chart of that schedule, the diagonal from the upper left corner of the rectangle of the first job in the schedule to the lower right corner of the rectangle of the last job in S_ℓ is above the diagonal from $(0, w(N))$ to $(p(N), 0)$. This holds for all $\ell \in [n]$. So, the area underneath the upper envelope of the rectangles, which equals the optimal objective value, is at least $\frac{1}{2} w(N)\, p(N)$, which corresponds to the area of the triangle that is defined by the diagonal from $(0, w(N))$ to $(p(N), 0)$ and the coordinate axes. The claim follows from the observation that any feasible schedule has an objective value $\leq w(N)\, p(N)$. □

Proposition 3.5 together with the algorithm in [108] to compute a density-maximizing initial set yields a 2-approximation algorithm for $1\,|\,prec|\,\sum w_j C_j$. In Section 3.5, we use a similar density-maximizing Greedy idea to obtain approximation algorithms for OR-scheduling. The main difference of OR-scheduling compared to AND-scheduling is that a decomposition result similar to Sidney [152] does not exist. In particular, the example in [45], which demonstrates that the performance guarantee of the Greedy algorithm for MSSC is tight, shows that a Sidney-like decomposition does not necessarily yield an optimal solution, if the density-maximizing feasible starting set is not unique.

Feige, Lovász and Tetali [45] presented an instance of MSVC where, after each iteration of the Greedy algorithm, there are two density-maximizing sets in the remaining instance. These sets each comprise of a single job with unit processing time and all its successors, which have zero processing time. In every iteration, Greedy picks one of these sets, whereas an optimum solution would be to always pick the other set.

3.3 Bipartite OR-Scheduling is Hard

As already indicated, see, e.g., Figure 3.2, bipartite OR-scheduling generalizes several NP-hard problems, so it is certainly NP-hard. Theorem 3.6 strengthens the NP-hardness result of Johannes [93], who showed that $1 \,|\, \textit{or-prec}, p_j = 1 \,|\, \sum w_j C_j$ is strongly NP-hard.

Theorem 3.6
The following are strongly NP-hard: $1 \,|\, \textit{or-prec} = \textit{bipartite}, p_j \in \{0,1\} \,|\, \sum C_j$ *and* $1 \,|\, \textit{or-prec} = \textit{bipartite}, p_j = 1 \,|\, \sum w_j C_j$ *with* $w_j \in \{0,1\}$.

Proof. The reduction goes from Exact Cover which is known to be NP-complete [98]. Exact Cover remains strongly NP-hard even if all sets contain only three elements, see [53]. That is, the input of an instance of Exact Cover consists of a positive integer $\ell \in \mathbb{N}$, a finite set of elements $U = \{e_1, \dots, e_{3\ell}\}$ and a collection of subsets $\mathcal{R} \subseteq 2^U$ with $|R| = 3$ for all $R \in \mathcal{R}$. The task is to decide whether or not there is an exact cover for U, i.e., whether there is $\mathcal{T} \subseteq \mathcal{R}$ with $|\mathcal{T}| = \ell$ such that $U = \bigcup_{R \in \mathcal{T}} R$.

Consider an instance of Exact Cover and the following reduction to bipartite OR-scheduling. We introduce one job for every set in $\mathcal{R}$ (set-jobs) and one job for every element of U (element-jobs). The precedence graph is given by $G = (\mathcal{R} \cup U, E_\vee)$ with $E_\vee = \{(R, e) \in \mathcal{R} \times U \,|\, e \in R\}$.[17] Observe that G is bipartite and that each set-job has exactly three successors. The weights and processing times depend on the scheduling problem we want to reduce to:

(*i*) for $1 \,|\, \textit{or-prec} = \textit{bipartite}, p_j \in \{0,1\} \,|\, \sum C_j$, we set $p_R = 1$ for all $R \in \mathcal{R}$ and $p_e = 0$ for all $e \in U$, and
(*ii*) for $1 \,|\, \textit{or-prec} = \textit{bipartite}, p_j = 1 \,|\, \sum w_j C_j$, we set $w_R = 0$ for all $R \in \mathcal{R}$ and $w_e = 1$ for all $e \in U$.

Note that, for both bipartite OR-scheduling problems, the element-jobs are the only jobs with zero processing time or positive weight, respectively. So, an optimal schedule processes all successors of a set-job immediately after the set-job completes.

Suppose we are given a Yes-instance and let $\mathcal{T} = \{R_1, \dots, R_\ell\} \subseteq \mathcal{R}$ be an exact cover for U. A feasible single-machine schedule is to first schedule the set-job R_1 followed by its three successors (the elements in R_1), then the set-job R_2 followed by its

[17] Note that G is obtained from the covering graph of the instance by reversing all arc directions.

three successors (the elements in R_2), and so on. The objective value of this schedule for *(i)*, i.e., $1 \,|\, \textit{or-prec} = \textit{bipartite}, p_j \in \{0,1\} \,|\, \sum C_j$, is equal to

$$\sum_{R \in \mathcal{R}} C_R + \sum_{q=1}^{\ell} |R_q| \, C_{R_q} = \sum_{q=1}^{|\mathcal{R}|} q + 3 \sum_{q=1}^{\ell} q = \frac{|\mathcal{R}|(|\mathcal{R}|+1) + 3\ell(\ell+1)}{2}. \tag{3.2}$$

For *(ii)*, i.e., $1 \,|\, \textit{or-prec} = \textit{bipartite}, p_j = 1 \,|\, \sum w_j C_j$, the objective value equals

$$\sum_{e \in U} C_e = \sum_{q=1}^{\ell} \sum_{e \in R_q} C_e = \sum_{q=1}^{\ell} \Big((4q-2) + (4q-1) + 4q \Big) = 6\ell^2 + 3\ell. \tag{3.3}$$

These are, in fact, the lowest possible objective values of any feasible schedule, since every set-job is predecessor of exactly three element-jobs. That is, at most three element-jobs become available when a set-job completes. So, any schedule with the above objective values, (3.2) or (3.3), starts with ℓ contiguous and disjoint "blocks" of the form $R_q \to e_{q_1} \to e_{q_2} \to e_{q_3}$ with $R_q = \{e_{q_1}, e_{q_2}, e_{q_3}\} \in \mathcal{R}$ for some $q \in [\ell]$. The sets corresponding to these ℓ set-jobs then form an exact cover for U. Hence, if we could solve $1 \,|\, \textit{or-prec} = \textit{bipartite}, p_j \in \{0,1\} \,|\, \sum C_j$ or $1 \,|\, \textit{or-prec} = \textit{bipartite}, p_j = 1 \,|\, \sum w_j C_j$ with $w_j \in \{0,1\}$ in polynomial time, then we could decide whether or not a given instance of Exact Cover is a Yes-instance. □

Note that the second problem in Theorem 3.6 is a special case of the problem considered in [93], and that $1 \,|\, \textit{or-prec}, p_j = 1 \,|\, \sum C_j$ is trivial.

3.4 Algorithms for Laminar Min-Sum Covering Problems

In the previous section, we observed that bipartite OR-scheduling is already strongly NP-hard for unit processing times and 0/1 weights, or vice versa. Thus, restriction of the processing times or weights does not seem to make OR-scheduling "significantly easier" from a complexity perspective. Another possibility to simplify bipartite OR-scheduling is to restrict the precedence relation further. For min-sum covering problems, this results in restricting the collection of subsets $\mathcal{R} \subseteq 2^U$. One typical special case of set cover problems is to restrict $\mathcal{R}$ to be laminar, see Definition 3.1.

We show that laminar MSSC and laminar all-but-K MSSC can be solved to optimality in polynomial time, and then propose a 2-approximation algorithm for laminar generalized min-sum set cover in Section 3.4.2. Recall that MSSC is a special case of all-but-K MSSC, i.e., any algorithm for the latter problem also solves MSSC. Before we turn to the more involved algorithm for laminar all-but-K MSSC in Section 3.4.1, we show that the natural Greedy algorithm for MSSC computes an optimal solution if the sets in $\mathcal{R}$ are laminar. We consider the more general *weighted laminar MSSC*, where each set in $\mathcal{R}$ is associated with a positive weight and the task is to minimize the sum of weighted covering times. That is, in each step, the Greedy algorithm picks the element in U that covers a subset of $\mathcal{R}$ of maximum total weight next.

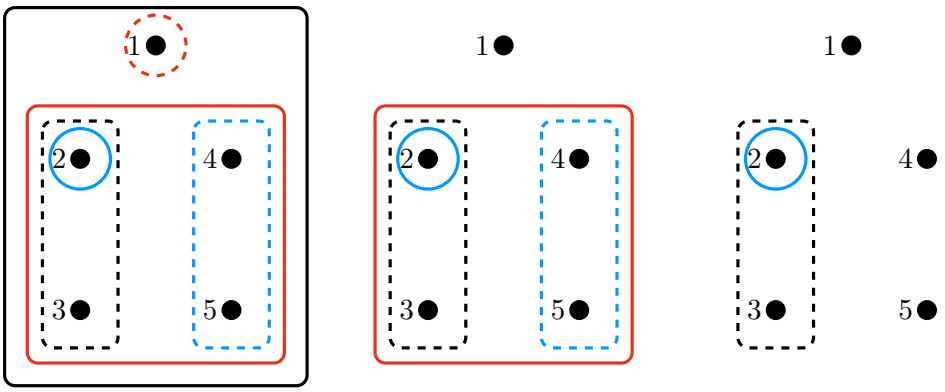

Fig. 3.4: The instance of Figure 3.1 with unit weights (left) and the uncovered sets after times 1 (middle) and 2 (right) for the ordering $1 \to 4 \to 5 \to 2 \to 3$. The objective value of this ordering is 14, and an optimal ordering for this instance is, e.g., $2 \to 1 \to 5 \to 4 \to 3$ with an objective value of 9. With $e = 2$, we get that $\mathcal{R}_e^+ = \{\{2\}, \{2,3\}\}$, $\mathcal{R}_e^- = \{\{2,3,4,5\}, \{1,2,3,4,5\}\}$ and $\overline{R} = \{2,3,4,5\}$. Hence, $f = 4$ and $\mathcal{R}_f^+ = \{\{4,5\}\}$.

Lemma 3.7
The Greedy algorithm is optimal for weighted laminar min-sum set cover.

Proof. W.l.o.g., we assume that $\mathcal{R}$ contains no duplicate sets. Otherwise, we could increase the weight $w_R \in \mathbb{N}$ of a set $R \in \mathcal{R}$ correspondingly and remove its copies. For $f \in U$, let $\mathcal{R}_f = \{R \in \mathcal{R} \,|\, f \in R\}$ be the sets that contain f, and let $e \in U$ be an element that the Greedy algorithm could pick first, i.e., $w(\mathcal{R}_e) \geq w(\mathcal{R}_f)$ for all $f \in U$. Consider any optimal linear ordering $\pi : U \to [|U|]$, and suppose that e is not the first element, i.e., $\pi(e) > 1$. We give a procedure of how to alter the optimal solution without increasing the objective function value such that e appears first in the linear ordering. We can iteratively repeat this argument for the other elements until the optimal solution coincides with the Greedy solution. Since the optimal and Greedy solution were chosen arbitrarily, this proves the claim.

Let $\mathcal{R}_e^+ = \{R \in \mathcal{R}_e \,|\, \pi(R) = \pi(e)\}$ be the sets that are covered by e, and let $\mathcal{R}_e^- = \mathcal{R}_e \setminus \mathcal{R}_e^+$ be the sets that were covered by previous elements in π. If $\mathcal{R}_e^- = \emptyset$, then let $f \in U$ be the element with $\pi(f) = 1$. Else, since $\mathcal{R}$ is laminar and the sets in $\mathcal{R}_e^-$ are covered before time $\pi(e)$, we get that $S \supseteq R$ for all $S \in \mathcal{R}_e^-$ and $R \in \mathcal{R}_e^+$. In particular, there is an inclusion-minimal set $\overline{R} \in \mathcal{R}_e^-$ that was covered by some element $f \in U$ with $\pi(f) < \pi(e)$, i.e., $\pi(\overline{R}) = \pi(f)$. Observe that $f \in \overline{R} \subseteq S$ for all $S \in \mathcal{R}_e^-$ implies $\pi(S) \leq \pi(f)$ for all $S \in \mathcal{R}_e^-$ and, further, $\mathcal{R}_e^- \subseteq \mathcal{R}_f$. So, f covers the last uncovered set(s) in $\mathcal{R}_e^-$, and we can write $\mathcal{R}_f = \mathcal{R}_e^- \dot\cup \mathcal{R}_f^+$.[18] Thus, $w(\mathcal{R}_e) \geq w(\mathcal{R}_f)$ implies $w(\mathcal{R}_e^+) \geq w(\mathcal{R}_f^+)$. Figure 3.4 illustrates the respective sets in a suboptimal ordering for the instance of Figure 3.1.

[18] In contrast to the definition of $\mathcal{R}_e^+$, the set $\mathcal{R}_f^+$ might contain sets that are covered before time $\pi(f)$.

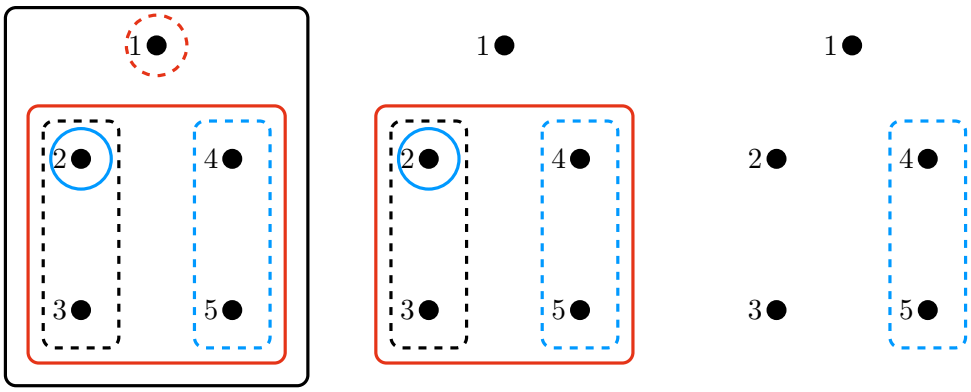

Fig. 3.5: The instance of Figure 3.4 (left), and the uncovered sets after time 1 (middle) and time 2 (right) for the ordering $1 \to 2 \to 5 \to 4 \to 3$ after swapping 2 and 4 in the linear ordering in Figure 3.4. The objective value of this ordering is 11. In the next iteration, we choose $e = 2$ and $f = 1$. The corresponding sets are then $\mathcal{R}_e^+ = \{\{2\}, \{2,3\}, \{2,3,4,5\}\}$, $\mathcal{R}_e^- = \{\overline{R}\}$ with $\overline{R} = \{1,2,3,4,5\}$ and $\mathcal{R}_f^+ = \{\{1\}\}$. The resulting ordering $2 \to 1 \to 5 \to 4 \to 3$ after swapping 2 and 1 is indeed optimal.

Let $S = \{f' \in U \mid \pi(f) < \pi(f') < \pi(e)\}$ be the elements that appear between f and e in the linear ordering. We swap f and e, i.e., we consider the linear ordering $\pi' : U \to [|U|]$ with $\pi'(e) = \pi(f)$, $\pi'(f) = \pi(e)$ and $\pi'(f') = \pi(f')$ for all $f' \in U \setminus \{e, f\}$. Compared to π, the covering time in π' of all sets in $\mathcal{R}_e^+$ decreases by $|S| + 1$, sets in $\mathcal{R}_f^+$ are covered at most $|S| + 1$ time steps later, and the covering times of all other sets remain unchanged. Figure 3.5 illustrates the resulting linear ordering after swapping e and f in the ordering in Figure 3.4. The difference in the objective values of π and π' is

$$\sum_{R \in \mathcal{R}} w_R \left(\pi'(R) - \pi(R)\right) \leq (|S| + 1) \left(w(\mathcal{R}_f^+) - w(\mathcal{R}_e^+)\right) \leq 0. \tag{3.4}$$

So, swapping e and f in the linear ordering does not increase the objective value, and e appears earlier in the new linear ordering. We can repeat this procedure iteratively until $\pi'(e) = 1$. (Note that the set $\mathcal{R}_e^+$ is strictly larger in the next iteration.) □

Recall the min-sum set cover instance in Example 1.12, see also Figure 1.3. Note that $\mathcal{R}$ is not laminar. For unit weights, the Greedy algorithm could choose the linear ordering $4 \to 3 \to 2 \to 1 \to 5$ with an objective value of 10. The optimal objective value for this instance is 9, i.e., Greedy is not optimal.

Restricting to laminar sets does simplify min-sum set cover, but the Greedy algorithm is not necessarily optimal for laminar generalized min-sum set cover with arbitrary covering requirements. Consider an instance with $U = \{1,2,3\}$ and $\mathcal{R} = \{\{1\}, \{2\}, \{2,3\}\}$. The covering requirements and the weights of the sets are equal to their cardinality, i.e., this is an instance of weighted laminar MLSC. Figure 3.6 illustrates the instance.

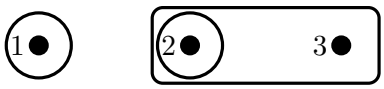

Fig. 3.6: An instance of weighted laminar MLSC with sets depicted in black. For weights $w_R = |R|$ for all $R \in \{\{1\}, \{2\}, \{2,3\}\}$, an optimum solution is to order the elements $2 \to 3 \to 1$ with an objective value of 8. For weights $\overline{w}_{\{1\}} = \overline{w}_{\{2,3\}} = 2$ and $\overline{w}_{\{2\}} = 1$, an optimum solution is $1 \to 2 \to 3$ with an objective value of 10.

An optimal solution is to order the elements $2 \to 3 \to 1$. The objective value of this ordering is $1 + 2 \cdot 2 + 3 = 8$. Since elements 1 and 2 both cover a set of weight 1, the Greedy algorithm can choose either of the two elements. By perturbing the weights slightly, we may assume that the Greedy algorithm chooses 1 first. In the next iteration, the algorithm is left with elements 2 (which covers a set of weight 1) and 3 (which does not cover a set). Thus, the Greedy algorithm returns the solution $1 \to 2 \to 3$ with an objective value of $1 + 2 + 2 \cdot 3 = 9$. Hence, the solution returned by the Greedy algorithm is not optimal.

In this example, it seems natural to prefer element 2 over 1, since element 2 not only covers the set $\{2\}$, but also "partially covers" $\{2, 3\}$. One idea to incorporate this intuition into the Greedy algorithm is to choose the element $e \in U$ that maximizes the partially covered total weight $\sum_{R \ni e} \frac{w_R}{\kappa(R)}$ next. The summand of each set is the ratio of its weight to the number of elements needed to cover the set. After each iteration, the covering requirements of all sets that contain the chosen element have to be decremented.

However, even this more intuitive Greedy algorithm can be tricked by adapting the weights in the above example slightly. Consider again the instance in Figure 3.6, but with new weights $\overline{w}_{\{1\}} = \overline{w}_{\{2,3\}} = 2$ and $\overline{w}_{\{2\}} = 1$. In the first iteration, the variant of the Greedy algorithm can again choose element 1 or 2, since $\frac{\overline{w}_{\{1\}}}{1} = 2 = \frac{\overline{w}_{\{2\}}}{1} + \frac{\overline{w}_{\{2,3\}}}{2}$. Suppose element 2 is chosen first. In the next step, the corresponding ratios are $\frac{\overline{w}_{\{1\}}}{1} = 2 = \frac{\overline{w}_{\{2,3\}}}{1}$. So, the Greedy can break ties arbitrarily and might return the ordering $2 \to 3 \to 1$ with an objective value of $1 + 2 \cdot 2 + 2 \cdot 3 = 11$. With these weights, however, an optimum solution is $1 \to 2 \to 3$ with an objective value of $2 + 2 + 2 \cdot 3 = 10$. So, it seems, one needs an algorithm that is more involved than a standard Greedy approach to tackle laminar generalized min-sum set cover.

3.4.1 Laminar All-But-Constant Min-Sum Set Cover

We now turn to the special case of laminar all-but-K min-sum set cover. That is, the collection of sets $\mathcal{R}$ is laminar (Definition 3.1), and the covering requirements are of the form $\kappa(R) = \max\{1, |R| - K\}$ for all $R \in \mathcal{R}$ and some $K \in \mathbb{N}_0$ (Definition 3.3). Since $\mathcal{R}$ is laminar, there is a unique inclusion-minimal superset for every $e \in U$ and every $R \in \mathcal{R}$, unless it is not contained in any other set. For $R \in \mathcal{R}$, we denote the set of elements that are not contained in any subset $S \in \mathcal{R}$ of R by $F_R := \{e \in R \mid \nexists\, S \in \mathcal{R} \text{ with } e \in S \subsetneq R\}$.

The elements in F_R are called *free in* R. A set $S \in \mathcal{R}$ with $S \subsetneq R$ is called *maximal in* R *w.r.t.* $\mathcal{R}$ if there is no $S' \in \mathcal{R}$ with $S \subsetneq S' \subsetneq R$.[19] We can partition a set $R \in \mathcal{R}$ into a disjoint union

$$R = F_R \cup \bigcup_{\substack{S \in \mathcal{R},\, S \subsetneq R \\ \text{max w.r.t. } \mathcal{R}}} S. \tag{3.5}$$

The idea of the algorithm to solve laminar all-but-K min-sum set cover is as follows: First, we construct an equivalent instance of laminar MLSC (Algorithm 2 and Lemmas 3.9 and 3.10), and then we solve this instance to optimality (Theorem 3.11). We actually consider the more general *weighted laminar all-but-K min-sum set cover*, where every set $R \in \mathcal{R}$ is associated with a positive weight $w_R \in \mathbb{N}$. To distinguish an instance of all-but-K MSSC from its equivalent MLSC instance, the sets in the MLSC instance are indicated with an overline, i.e., are denoted by $\overline{R}$, $\overline{S}$, etc. Similarly, we denote the collection of sets of the MLSC instance by $\overline{\mathcal{R}}$. To describe the algorithm, we need the notion of a *level* of a set.

Definition 3.8 (Level of a Set)
Let $\mathcal{R}$ be laminar, $R \in \mathcal{R}$ and $F_R \subseteq R$ be the free elements in R. The *level of* R is recursively defined as

(*i*) $\operatorname{lev}(R) \coloneqq 0$ if $R = F_R$, i.e., there is no $S \in \mathcal{R}$ with $S \subsetneq R$, and
(*ii*) $\operatorname{lev}(R) \coloneqq 1 + \max\{\operatorname{lev}(S) \mid S \in \mathcal{R},\, S \subsetneq R \text{ maximal in } R \text{ w.r.t. } \mathcal{R}\}$.

The algorithm to transform an instance of all-but-K MSSC to an equivalent instance of MLSC works as follows: We iterate over the levels, i.e., we start with the sets in $\mathcal{R}$ at level 0, which do not contain any subset in $\mathcal{R}$. Since the elements in U are indistinguishable, it does not matter which $\kappa(R)$ of the $|R|$ elements cover a set $R \in \mathcal{R}$. We remove $|R| - \kappa(R)$ of the free elements from any set R at level 0. Note that these elements are not removed from the instance, but are now free in the unique inclusion-minimal superset of R. The resulting set, which is a subset of R, is denoted by $\overline{R}$. We continue in the same manner with the sets at levels $1, \ldots, |U|$, i.e., we remove $|R| - \kappa(R)$ of the free elements in R. If there are not enough free elements to be removed, we additionally remove the maximal (in R w.r.t. $\overline{\mathcal{R}}$) singleton sets $\overline{\{e\}} \in \overline{\mathcal{R}}$ with $e \in R$ of lowest weight from R, and "place" them into the inclusion-minimal superset of R instead. Lemma 3.9 states that this can be done in a feasible way.

Thereby, we successively decrease the size of all sets in $\mathcal{R}$ until their cardinalities coincide with their covering requirements. This procedure might produce duplicate sets if we create the set $\overline{R}$ when dealing with a set $R \in \mathcal{R}$, although the collection $\overline{\mathcal{R}}$ already contains a set $\overline{S} = \overline{R}$ from an earlier iteration where $S \in \mathcal{R}$ was considered. In this case, the "old set" $\overline{S} \in \overline{\mathcal{R}}$ is replaced by the "new set" $\overline{R}$, i.e., we update $\overline{\mathcal{R}}$.

[19] Note that $S \subsetneq R$ with $S \in \mathcal{R}$ can be maximal in R w.r.t. $\mathcal{R}$, although R is not contained in $\mathcal{R}$.

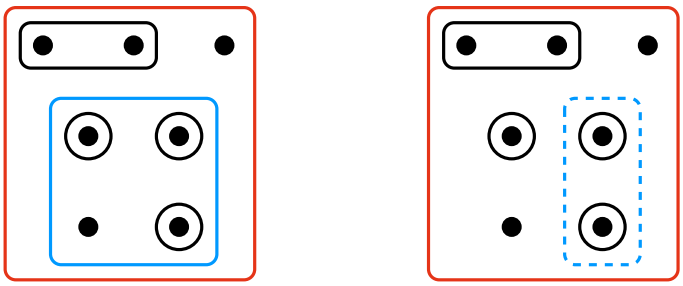

Fig. 3.7: An iteration of Algorithm 2 for $K = 2$. On the left, the current set R and its inclusion-minimal superset are depicted in blue and red, respectively. All other sets are depicted in black. Since $|F_R| = 1 < 2 = |R| - K = \kappa(R)$, also a singleton is removed from R. The resulting set $\overline{R}$ is depicted in blue dashed lines on the right.

The weight of $\overline{R}$ is the sum of the weights of $\overline{S}$ and R. Note that this case only occurs if R is a superset of S, i.e., if $\text{lev}(R) > \text{lev}(S)$. By updating $\overline{\mathcal{R}}$ in this particular way, we ensure that $\overline{S} \subsetneq R$ is maximal in R w.r.t. $\overline{\mathcal{R}}$ if and only if the corresponding set $S \in \mathcal{R}$ with $S \subsetneq R$ is maximal in R w.r.t. $\mathcal{R}$. Also, if $\overline{R} = \{e\}$ is a singleton, then $e \in S \subseteq R$ for all subsets $S \in \mathcal{R}$ of $R \in \mathcal{R}$.

The algorithm is summarized in Algorithm 2. The free elements and singleton sets in the current set R are defined in lines 4 and 5, respectively. The if-clause in line 6 checks whether there are "enough" free elements to be removed. If not, the algorithm chooses all free elements and the singleton sets of lowest weight to be removed in line 11. Finally, the if-clause in line 14 checks whether a set is duplicated, and if so, the "old set" is replaced by the "new one" and the weights are updated accordingly. Note that Algorithm 2 runs in polynomial time, since every set in $\mathcal{R}$ is considered exactly once. Figure 3.7 illustrates an iteration of the algorithm. The following lemma shows that there are always enough free elements and singletons to be removed, and that the output of Algorithm 2 is an instance of weighted laminar minimum latency set cover.

Lemma 3.9

Algorithm 2 works correctly, i.e., for every $R \in \mathcal{R}$, we have $|R| \leq |F| + |\mathcal{S}| + \kappa(R)$, where F are the free elements and $\mathcal{S}$ are the singletons in the iteration of R. Further, the resulting collection $\overline{\mathcal{R}}$ is laminar.

Proof. We prove the first part of the statement by induction on the level. If $\text{lev}(R) = 0$ then $R = F$, i.e., $|R| \leq |F| + \kappa(R)$ by non-negativity of $\kappa(R)$. Let $\ell \in [|U|]$ and consider a set with $\text{lev}(R) = \ell$. By induction hypothesis, all sets $S \in \mathcal{R}$ with $\text{lev}(S) \leq \ell - 1$ were already processed by the algorithm in previous iterations. Note that $|\overline{S}| = \kappa(S) \leq |S|$ for $S \in \mathcal{R}$ and its corresponding set $\overline{S} \in \overline{\mathcal{R}}$. Similar to (3.5), we get

$$R = F \cup \bigcup_{\substack{\overline{S} \in \overline{\mathcal{R}},\, \overline{S} \subsetneq R \\ \text{max w.r.t. } \overline{\mathcal{R}}}} \overline{S} = F \cup \bigcup_{\{e\} \in \mathcal{S}} \{e\} \cup \bigcup_{\substack{\overline{S} \in \overline{\mathcal{R}},\, \overline{S} \subsetneq R \\ \text{max w.r.t. } \overline{\mathcal{R}} \\ |\overline{S}| \geq 2}} \overline{S}. \tag{3.6}$$

Input: Instance $(U, \mathcal{R}, \kappa)$ of weighted laminar all-but-K MSSC
Output: Instance $(U, \overline{\mathcal{R}}, \overline{\kappa})$ of weighted laminar MLSC

$\overline{\mathcal{R}} \leftarrow \emptyset$;
for $\ell = 0, 1, \ldots, |U|$ **do**
  for $R \in \mathcal{R}$ *with* $\operatorname{lev}(R) = \ell$ **do**
    $F \leftarrow R \setminus \bigcup \{\overline{S} \,|\, \overline{S} \in \overline{\mathcal{R}},\, \overline{S} \subsetneq R\}$;
    $\mathcal{S} \leftarrow \{\overline{\{e\}} \in \overline{\mathcal{R}} \,\big|\, \overline{\{e\}} \subsetneq R \text{ is maximal in } R \text{ w.r.t. } \overline{\mathcal{R}}\}$;
    if $|F| \geq |R| - \kappa(R)$ **then**
      Choose $F^- \subseteq F$ of cardinality $|R| - \kappa(R)$;
    else
      Sort $\mathcal{S}$ in non-decreasing order of the weights;
      Let $\mathcal{S}^- \subseteq \mathcal{S}$ be the first $|R| - \kappa(R) - |F|$ elements of $\mathcal{S}$;
      $F^- \leftarrow F \cup \{e \,|\, \overline{\{e\}} \in \mathcal{S}^-\}$;
    end
    $\overline{R} \leftarrow R \setminus F^-$;
    if $\exists\, \overline{S} \in \overline{\mathcal{R}}$ *with* $\overline{S} = \overline{R}$ **then**
      $\overline{\mathcal{R}} \leftarrow (\overline{\mathcal{R}} \setminus \{\overline{S}\}) \cup \{\overline{R}\}$ and $w_{\overline{R}} \leftarrow w_{\overline{S}} + w_R$;
    else
      $\overline{\mathcal{R}} \leftarrow \overline{\mathcal{R}} \cup \{\overline{R}\}$ and $w_{\overline{R}} \leftarrow w_R$;
    end
  end
end
return instance $(U, \overline{\mathcal{R}}, \overline{\kappa})$ with requirements $\overline{\kappa} : \overline{\mathcal{R}} \to \mathbb{N}$, $\overline{\kappa}(\overline{R}) = |\overline{R}|$;

Algorithm 2: An algorithm to transform an instance of weighted laminar all-but-K MSSC into an equivalent instance of weighted laminar MLSC.

If R contains no maximal subsets w.r.t. $\overline{\mathcal{R}}$ of cardinality ≥ 2, then $|R| = |F| + |\mathcal{S}| \leq |F| + |\mathcal{S}| + \kappa(R)$. Further, $\kappa(R) = 1$ implies $\kappa(S) = 1$ for all $S \subsetneq R$, $S \in \mathcal{R}$. In that case, all sets $\overline{S} \in \overline{\mathcal{R}}$ with $\overline{S} \subsetneq R$ are singletons. So, assume there is at least one maximal subset w.r.t. $\overline{\mathcal{R}}$ of cardinality ≥ 2 and $\kappa(R) = |R| - K > 1$. With (3.6), we obtain

$$\begin{aligned} |R| &= |F| + |\mathcal{S}| + \sum_{\substack{\overline{S} \in \overline{\mathcal{R}},\, \overline{S} \subsetneq R \\ \text{max w.r.t. } \overline{\mathcal{R}} \\ |\overline{S}| \geq 2}} |\overline{S}| = |F| + |\mathcal{S}| + \sum_{\substack{S \in \mathcal{R},\, S \subsetneq R \\ \text{max w.r.t. } \mathcal{R} \\ |S| \geq K+2}} \kappa(S) \\ &= |F| + |\mathcal{S}| + \sum_{\substack{S \in \mathcal{R},\, S \subsetneq R \\ \text{max w.r.t. } \mathcal{R} \\ |S| \geq K+2}} (|S| - K) \leq |F| + |\mathcal{S}| + |R| - K \\ &= |F| + |\mathcal{S}| + \kappa(R). \end{aligned} \tag{3.7}$$

As for the second part of the statement, let $\overline{\mathcal{R}}_\ell$ be the current collection of subsets at the end of the for-loop for $\ell \in \{0, 1, \dots, |U|\}$. Clearly, $\overline{\mathcal{R}}_0$ is laminar, since it comprises disjoint sets. Now, assume that $\overline{\mathcal{R}}_{\ell-1}$ is laminar for some $\ell \in [|U|]$. In the next iteration of the outer for-loop, the set $\overline{R}$ is derived from $R \in \mathcal{R}$ with $\operatorname{lev}(R) = \ell$ by removing free elements and singletons. Note that $\overline{R} \in \overline{\mathcal{R}}_\ell$ is a superset of all non-singleton sets $\overline{S} \in \overline{\mathcal{R}}_{\ell-1}$ with $S \subsetneq R$, $S \in \mathcal{R}$. By induction, $\overline{\mathcal{R}}_{\ell-1} \cup \{\overline{R}\}$ is laminar. Also, since $\mathcal{R}$ is laminar, two sets $R, R' \in \mathcal{R}$ with $\operatorname{lev}(R) = \operatorname{lev}(R') = \ell$ do not intersect by Definition 3.8. Hence, the resulting sets $\overline{R}, \overline{R'} \in \overline{\mathcal{R}}_\ell$ do not intersect either. So, $\overline{\mathcal{R}}_\ell$ is laminar and, therefore, $\overline{\mathcal{R}} = \overline{\mathcal{R}}_{|U|}$ is laminar as well. This proves the claim. □

Lemma 3.10
Let $\mathcal{I}$ be an instance of weighted laminar all-but-K min-sum set cover, and let $\overline{\mathcal{I}}$ be the corresponding instance of weighted laminar minimum latency set cover that is returned by Algorithm 2. Then any optimal solution for $\overline{\mathcal{I}}$ is also optimal for $\mathcal{I}$.

Proof. Consider a linear ordering $\pi : U \to [|U|]$ that is optimal for $\mathcal{I}$. We show that we can alter π without increasing its objective value such that it is also feasible for $\overline{\mathcal{I}}$. So, any optimal solution for $\overline{\mathcal{I}}$ has lower total weighted covering time than π. This proves the claim, since any feasible solution for $\overline{\mathcal{I}}$ is also feasible for $\mathcal{I}$.

For $R \in \mathcal{R}$, let $\overline{F}_R$ and $\mathcal{S}_R$ be the free elements and singletons in the corresponding iteration of Algorithm 2, respectively. Let $\mathcal{S}'_R = \{S \in \mathcal{R} \mid S \subsetneq R, \kappa(S) = 1\}$ be the subsets of R that correspond to the singletons in $\mathcal{S}_R$. That is, $\mathcal{S}'_R$ is the set of "pre-images" of the singletons in $\mathcal{S}_R$. Recall that every set R is covered as soon as $\kappa(R)$ elements appear in the linear ordering. In particular, $\pi(e) \geq \pi(S')$ for all $\overline{\{e\}} \in \mathcal{S}_R$ and $S' \in \mathcal{S}'_R$ with $e \in S'$. We show by induction on the level that, w.l.o.g., the following holds for all $R \in \mathcal{R}$:

(*i*) $\pi(S) \leq \pi(R)$ for all $S \subseteq R$, $S \in \mathcal{R}$ with $\kappa(S) > 1$,
(*ii*) $\pi(e) > \pi(R)$ for all $e \in \overline{F}_R \setminus \overline{R}$, and
(*iii*) if $\pi(f) \leq \pi(R) < \pi(e)$ for $e, f \in R$ with $\overline{\{e\}}, \overline{\{f\}} \in \mathcal{S}_R$, then $w_{\overline{\{f\}}} \geq w_{\overline{\{e\}}}$.

Observe that π is feasible for the MLSC instance $\overline{\mathcal{I}}$ if it satisfies these three conditions. The first condition yields that the subsets with non-unit requirement of every set have no higher covering time than the set itself. Conditions (*ii*) and (*iii*) state that a set is not preceded by the elements and singletons that were removed by Algorithm 2. It remains to be shown that we can alter the optimal solution π in such a way that it satisfies (*i*), (*ii*) and (*iii*).

For $\operatorname{lev}(R) = 0$, (*i*) and (*iii*) are trivially true. As for (*ii*), suppose there is $e \in \overline{F}_R \setminus \overline{R}$ with $\pi(e) \leq \pi(R)$. The set R is covered as soon as $\kappa(R)$ elements have appeared, so there is $f \in \overline{R}$ with $\pi(R) < \pi(f)$. Recall that $\operatorname{lev}(R) = 0$ implies that $\mathcal{R}$ does not contain any subsets of R. Hence, e and f are contained in the same sets in $\mathcal{R}$.

So, swapping e and f, i.e., setting $\pi'(e) = \pi(f)$ and $\pi'(f) = \pi(e)$, does neither destroy feasibility of the linear ordering nor does the objective value increase. This establishes the base case. Now, let $R \in \mathcal{R}$ with $\text{lev}(R) \geq 1$, i.e., $\mathcal{R}$ contains at least one subset of R.

(*i*) For $\kappa(R) = 1$, the statement is trivially true because then $\kappa(S) = 1$ holds for all $S \subsetneq R$. So, assume $\kappa(R) = |R| - K$ and suppose a set $S \subsetneq R$ with $\kappa(S) = |S| - K$ is covered after R, i.e., $\pi(S) > \pi(R)$. That is, K elements of S appear strictly after time $\pi(S)$. Since S is covered as soon as $\kappa(S)$ elements of S have appeared, there is an element in $S \subseteq R$ that appears at time $\pi(S) > \pi(R)$. Hence, at least $K + 1$ elements of R appear strictly after time $\pi(R)$. This is a contradiction to the feasibility of the solution, because then at most $|R| - (K + 1) < \kappa(R)$ elements of R appear before time $\pi(R)$.

(*ii*) Suppose there is an element $e \in \overline{F}_R \setminus \overline{R}$ with $\pi(e) \leq \pi(R)$. Similar to the base case, there is $f \in \overline{R}$ with $\pi(R) < \pi(f)$. If there is a set $S \subsetneq R$, $S \in \mathcal{R}$ with $e \in S$, then $e \in \overline{F}_R$ implies that e was also part of the free elements in the iteration of S, i.e., $e \in \overline{F}_S \setminus \overline{S}$. By induction ($\text{lev}(S) < \text{lev}(R)$), we get that $\pi(e) > \pi(S)$. So, e does not contribute to the covering of any subset of R, but only to supersets $R' \supseteq R$ with $R' \in \mathcal{R}$. But then $f \in R \subseteq R'$ implies that we can swap e and f without loosing feasibility or increasing the objective value (some sets that contain f might be even covered earlier).

(*iii*) Suppose there are $e, f \in R$ with $\overline{\{e\}}, \overline{\{f\}} \in \mathcal{S}_R$ and $\pi(f) \leq \pi(R) < \pi(e)$ such that $w_{\overline{\{f\}}} < w_{\overline{\{e\}}}$. The sets in $\mathcal{S}'_R$ that contain e or f have a lower level than R. By induction and (*ii*), all sets in $\mathcal{S}'_R$ that contain e or f are covered at time $\pi(e)$ or $\pi(f)$, respectively. Since $\overline{\{e\}}, \overline{\{f\}} \in \mathcal{S}_R$ are maximal in R w.r.t. $\overline{\mathcal{R}}$, the corresponding sets $S_e, S_f \in \mathcal{R}$ with $\overline{S_e} = \overline{\{e\}}$ and $\overline{S_f} = \overline{\{f\}}$ are maximal in R w.r.t. $\mathcal{R}$. Hence, there is no $S' \subsetneq R$, $S' \in \mathcal{R}$ with $S_e \subsetneq S' \subsetneq R$ or $S_f \subsetneq S' \subsetneq R$. That is, e and f do not contribute to the covering of any subset of R other than the subsets of S_e and S_f, respectively. Similar to (*ii*) above, we obtain a feasible linear ordering if we swap e and f (together with the corresponding sets in $\mathcal{S}'_R$). Further, this new linear ordering has a strictly better objective value, since $w_{\overline{\{f\}}} < w_{\overline{\{e\}}}$, similar to (3.4). This contradicts to the optimality of π.

Thus, we can assume that the optimal solution $\pi : U \to [|U|]$ satisfies (*i*), (*ii*) and (*iii*). That is, π is feasible for the MLSC instance $\overline{\mathcal{I}}$. Hence, any optimal solution for $\overline{\mathcal{I}}$ is also optimal for the initial instance $\mathcal{I}$ of all-but-K MSSC. □

Lemma 3.9 shows that the instance returned by Algorithm 2 is indeed an instance of laminar minimum latency set cover, and Lemma 3.10 states that solving this instance is equivalent to solving the initial instance of laminar all-but-K min-sum set cover. We are now in the position of proving the main result of this section.

Theorem 3.11
Weighted laminar all-but-K min-sum set cover can be solved in polynomial time.

Proof. Consider an instance $\mathcal{I}$ of weighted laminar all-but-K MSSC. First, we apply Algorithm 2, which runs in polynomial time, to obtain the corresponding instance $\overline{\mathcal{I}}$ of weighted laminar minimum latency set cover. By Lemma 3.10, any optimal solution for $\overline{\mathcal{I}}$ is also optimal for $\mathcal{I}$. It remains to be shown that we can solve weighted laminar MLSC in polynomial time.

Let $\overline{\mathcal{I}} = (U, \overline{\mathcal{R}}, \overline{\kappa})$ be an instance of weighted laminar MLSC. We construct an equivalent AND-scheduling instance by using the intree representation of $\overline{\mathcal{I}}$. That is, we introduce a job j_e for every element $e \in U$ and a job $j_{\overline{R}}$ for every set $\overline{R} \in \overline{\mathcal{R}}$. The processing times and weights of the jobs are $p_{j_e} = 1$, $w_{j_e} = 0$ for all $e \in U$ and $p_{j_{\overline{R}}} = 0$, $w_{j_{\overline{R}}} = w_{\overline{R}}$ for all $\overline{R} \in \overline{\mathcal{R}}$. The jobs $\{j_e \mid e \in U\}$ do not have any predecessors, and, for $\overline{R} \in \overline{\mathcal{R}}$, the predecessors of $j_{\overline{R}}$ are the jobs corresponding to the maximal subsets and the free elements in $\overline{R}$. That is, we introduce an arc $(j_{\overline{S}}, j_{\overline{R}}) \in E_\wedge$ in the precedence graph for all $\overline{S}, \overline{R} \in \overline{\mathcal{R}}$ if $\overline{S} \subsetneq \overline{R}$ is maximal in $\overline{R}$ w.r.t. $\overline{\mathcal{R}}$, and an arc $(j_e, j_{\overline{R}}) \in E_\wedge$ for every $\overline{R} \in \overline{\mathcal{R}}$ and free element $e \in F_{\overline{R}}$. Clearly, there is a one-to-one correspondence between a linear ordering of the elements in U and a feasible single-machine schedule of the jobs in $\{j_e \mid e \in U\}$. Also, finding a linear ordering that minimizes the sum of weighted covering times is equivalent to finding a schedule that minimizes the sum of weighted completion times.

Since $\overline{\mathcal{R}}$ is laminar, every set in $\overline{\mathcal{R}}$ is maximal in at most one set and every element is free in at most one set. So, every job of the scheduling instance described above has at most one successor, i.e., the constructed precedence graph $G = (N, E_\wedge)$ is an inforest, where $N = \{j_e \mid e \in U\} \cup \{j_{\overline{R}} \mid \overline{R} \in \overline{\mathcal{R}}\}$ is the set of jobs. Hence, we can solve weighted laminar MLSC by solving the corresponding scheduling problem $1 \mid prec \mid \sum w_j C_j$ with an inforest precedence graph, which can be done in polynomial time [85]. □

Note that the all-but-constant structure of the covering requirements is crucial for the inequality in (3.7) and, thus, for the correctness of Algorithm 2. In particular, the algorithm does not work if, for instance, a superset S of some set R satisfies $\kappa(S) < \kappa(R)$. Thus, it seems that one needs other ideas to show that laminar generalized min-sum set cover can be solved in polynomial time, in case it is contained in P at all.

3.4.2 Laminar Generalized Min-Sum Set Cover

Next, we present a 2-approximation algorithm for generalized min-sum set cover on a laminar family of sets $\mathcal{R}$. This result is achieved by combining an algorithm of Im, Sviridenko and Zwaan [88] for the preemptive variant of generalized min-sum set cover with the observation that there is no benefit in preemption if the sets are laminar. We first define *preemptive generalized min-sum set cover* and then discuss our algorithm and the connection to [88].

To illustrate preemptive GMSSC, it is convenient to interpret an instance of GMSSC as a scheduling problem with a job of unit processing time for every element and a job of zero processing time for every set together with the natural precedence relation defined by the covering graph. The job corresponding to set $R \in \mathcal{R}$ requires at least $\kappa(R)$ of its predecessors to be completed before it can start. If we assign a weight of zero to each element-job and a weight of one to each set-job, it becomes clear that the single-machine scheduling problem with the sum of weighted completion times objective is equivalent to generalized min-sum set cover.

Preemptive generalized min-sum set cover can be seen as the preemptive variant of this scheduling problem with the relaxed covering requirement that, for the job corresponding to $R \in \mathcal{R}$ to become available, a total processing load of at least $\kappa(R)$ of its predecessors needs to be completed. Formally, instead of defining a linear ordering, we denote the fraction of element $e \in U$ that is assigned to time $t \in [|U|]$ by $x_{et} \in [0;1]$. Certainly, each element needs to be completely assigned, i.e., $\sum_{t=1}^{|U|} x_{et} = 1$ for all $e \in U$, and a total fraction of one is assigned to each point in time, i.e., $\sum_{e \in U} x_{et} = 1$ for all $t \in [|U|]$. The covering time of a set $R \in \mathcal{R}$ is then defined as

$$\pi(R) \coloneqq \min \left\{ t \in [|U|] \;\middle|\; \sum_{e \in R} \sum_{\tau=1}^{t} x_{e\tau} \geq \kappa(R) \right\}. \tag{3.8}$$

Note that (3.8) coincides with (1.4) if all x_{et} are binary, i.e., define a linear ordering.

Im, Sviridenko and Zwaan [88] proposed a so-called *configuration LP* and showed that it is a valid relaxation for preemptive generalized min-sum set cover. This LP can be solved in polynomial time despite an exponential number of variables [88]. The authors use the concept of random α-points, which we discuss and also use in Chapter 4, to obtain a 2-approximate solution for preemptive GMSSC. Moreover, they present a procedure to transform any preemptive solution into a non-preemptive one. This transformation brings another factor of 6.2 in the approximation factor, which yields a 12.4-approximation for generalized min-sum set cover in total, see [88].

Our main result in this section is that, for laminar generalized min-sum set cover, any preemptive solution can be transformed into a non-preemptive solution without increasing the objective value. Thereby, the algorithm of [88] for preemptive GMSSC in combination with our transformation yields a 2-approximation algorithm for laminar generalized min-sum set cover.

As input, the algorithm to transform a preemptive solution for laminar generalized min-sum set cover into a non-preemptive one receives the covering times of the sets in a feasible preemptive solution. The sets are processed in an *earliest covering time first* manner (ties are broken arbitrarily). That is, in each iteration an element of an uncovered set with lowest covering time is scheduled. If the set R considered in the current iteration contains no uncovered subsets in $\mathcal{R}$, then the element is chosen arbitrarily within R (recall that the elements are indistinguishable). Else, the current set is updated to an uncovered subset of R of lowest covering time.

Input: Instance $(U, \mathcal{R}, \kappa)$ of laminar GMSSC and a feasible preemptive solution with covering times $\pi' : \mathcal{R} \to [|U|]$
Output: Feasible non-preemptive solution of laminar GMSSC in form of a linear ordering of the elements

```
1  S ← R and t ← 1;
2  while t ≤ |U| do
3  |  Choose R ∈ argmin{π'(S) | S ∈ S};
4  |  while ∃ S ∈ S with S ⊊ R do
5  |  |  Set R' ← R and choose R ∈ argmin{π'(S) | S ∈ S, S ⊊ R'};
6  |  end
7  |  Choose e ∈ R and set π(e) ← t;
8  |  for S ∈ S with e ∈ S do
9  |  |  κ(S) ← κ(S) − 1;
10 |  |  if κ(S) = 0 then
11 |  |  |  S ← S \ {S} and π(S) ← t;
12 |  |  else
13 |  |  |  S ← (S \ {S}) ∪ {S \ {e}};
14 |  |  end
15 |  |  t ← t + 1;
16 |  end
17 end
18 return π;
```

Algorithm 3: An algorithm to transform a preemptive solution of laminar generalized min-sum set cover into a non-preemptive one.

In this way, the element scheduled next is always chosen from the most urgent subset(s). Finally, after an element is scheduled, this element is removed from the instance and the covering requirements are updated accordingly. The covering time of a set is the first point in time when its remaining covering requirement is set to zero. The complete algorithm is summarized in Algorithm 3.

Note that Algorithm 3 returns a feasible (non-preemptive) solution for generalized min-sum set cover, since exactly one element is assigned to time t (line 7) in each iteration of the outer while-loop. If a set is covered, it is removed from the instance in line 11 and its covering time is set to t.[20] Else, the element is removed from the instance and the set is updated accordingly. Before we show that the covering times of the solution returned by Algorithm 3 satisfy $\pi(R) \leq \pi'(R)$ for all $R \in \mathcal{R}$, we illustrate the procedure using an example.

[20] Technically, one does not need to assign the covering time of a set in line 11, as the covering time is already well-defined by the linear ordering $\pi : U \to [|U|]$, see (1.4).

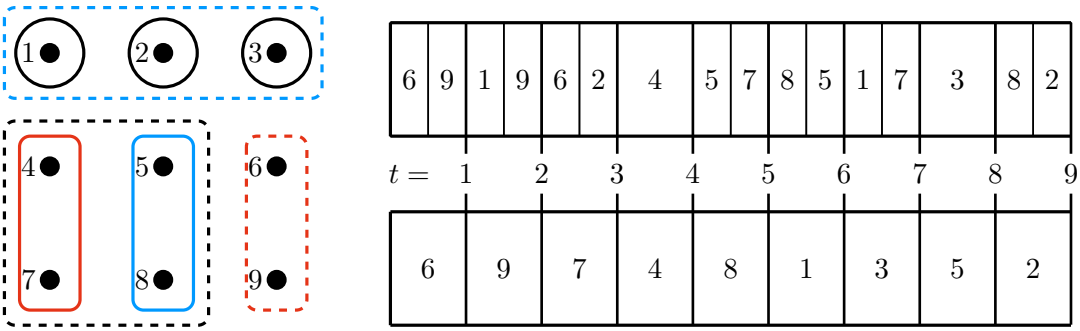

Fig. 3.8: The instance of Example 3.12 (left), and a feasible preemptive solution (top right) and the solution returned by Algorithm 3 (bottom right) illustrated as single-machine schedules. Note that the order of the elements within one time slot $[t-1;t]$ is arbitrary and the covering times are integers by definition, see (3.8).

Example 3.12 (Preemptive Laminar Generalized Min-Sum Set Cover)
Consider the instance depicted in Figure 3.8 (left) with $U = [9]$ and

$$\mathcal{R} = \{\{1\},\{2\},\{3\},\{1,2,3\},\{4,7\},\{5,8\},\{4,5,7,8\},\{6,9\}\}. \tag{3.9}$$

The covering requirements are $\kappa(\{1\}) = \kappa(\{2\}) = \kappa(\{3\}) = 1$, $\kappa(\{1,2,3\}) = \kappa(\{4,7\}) = \kappa(\{5,8\}) = \kappa(\{6,9\}) = 2$ and $\kappa(\{4,5,7,8\}) = 3$. Consider the preemptive solution on the top right of Figure 3.8. The covering times are $\pi'(\{6,9\}) = 3$, $\pi'(\{4,5,7,8\}) = 6$, $\pi'(\{1\}) = \pi'(\{4,7\}) = 7$, $\pi'(\{3\}) = \pi'(\{1,2,3\}) = 8$ and $\pi'(\{2\}) = \pi'(\{5,8\}) = 9$.

In the first iteration, the algorithm chooses $R = \{6,9\}$ and, since the set comprises free elements only, elements 6 and 9 are scheduled first. For $t = 3$, Algorithm 3 first picks $R = \{4,5,7,8\}$. Since this set contains uncovered subsets, the algorithm updates $R = \{4,7\}$ in the while-loop in line 5. Similar to before, the elements 4 and 7 are scheduled next. At time $t = 5$, the algorithm again chooses $R = \{4,5,7,8\}$ and updates $R = \{5,8\}$ in line 5 (note that $\{4,7\}$ is covered already). Since $\{5,8\}$ contains free elements only, an arbitrary element (in this case 8) is chosen, and $\{4,5,7,8\}$ is covered at time 5. The next uncovered set is $\{1\}$, so the algorithm chooses element 1 at time 6. Next, w.l.o.g., $\{1,2,3\}$ is chosen, which contains the still uncovered sets $\{2\}$ and $\{3\}$. Since $\pi'(\{3\}) < \pi'(\{2\})$, Algorithm 3 updates $R = \{3\}$ and schedules element 3. So, sets $\{3\}$ and $\{1,2,3\}$ are covered at time 7. Finally, the algorithm proceeds with the remaining sets $\{5,8\}$ and $\{2\}$ and schedules elements 5 and 2 last.

The covering times of the solution returned by Algorithm 3 are $\pi(\{6,9\}) = 2$, $\pi(\{4,7\}) = 4$, $\pi(\{4,5,7,8\}) = 5$, $\pi(\{1\}) = 6$, $\pi(\{3\}) = \pi(\{1,2,3\}) = 7$, $\pi(\{5,8\}) = 8$ and $\pi(\{2\}) = 9$. Note that $\pi(R) \leq \pi'(R)$ for all $R \in \mathcal{R}$.

Instead of breaking ties in lines 3 and 5 arbitrarily, we now assume that Algorithm 3 processes a set R completely in the following iterations of outer while-loop before it continues with a set disjoint from R. For convenience, we refrain from making this explicit in the algorithm.

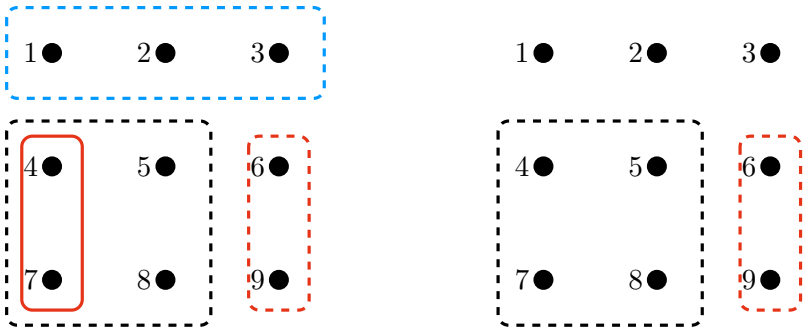

Fig. 3.9: The sets contained in $\mathcal{R}_R^- \cup \{R\}$ (left) and $\mathcal{D}_R$ (right) for $R = \{1, 2, 3\}$ in the instance of Example 3.12. Recall that $\pi'(\{1, 2, 3\}) = 8$, so $\mathcal{R}_R^- = \{\{6, 9\}, \{4, 5, 7, 8\}, \{4, 7\}\}$. The inclusion-maximal sets in $\mathcal{R}_R^-$ containing free elements are $\mathcal{D}_R = \{\{6, 9\}, \{4, 5, 7, 8\}\}$.

Theorem 3.13
Given a feasible preemptive solution for laminar generalized min-sum set cover with covering times $\pi' : \mathcal{R} \to [|U|]$, Algorithm 3 returns a feasible non-preemptive solution with covering times $\pi(R) \leq \pi'(R)$ for all $R \in \mathcal{R}$.

Proof. W.l.o.g., we assume that a set $R \in \mathcal{R}$ with $|R| > \kappa(R)$ contains no free elements, i.e., $F_R = \emptyset$ and any such set decomposes into subsets, see (3.5). Any "reasonable solution" for laminar GMSSC will never choose a free element in R if there is still an element in some subset of R available. That is, we can apply a preprocessing algorithm similar to Algorithm 2 that successively removes free elements from sets $R \in \mathcal{R}$ with $|R| > \kappa(R)$ and places those elements in the inclusion-minimal superset of R.

Let $R \in \mathcal{R}$ be the set chosen by Algorithm 3 after the last iteration of the inner while-loop (line 5) at time $t \in [|U|]$. We consider the instance reduced to the sets disjoint from R that are covered by time $\pi'(R)$ in the preemptive solution. Let $\mathcal{R}_R^- = \{S \in \mathcal{R} \mid \pi'(S) \leq \pi'(R), S \cap R = \emptyset\}$ be those sets. Further, we let $\mathcal{D}_R \subseteq \mathcal{R}_R^-$ be the set of inclusion-maximal sets in $\mathcal{R}_R^-$ that contain free elements. Note that $\mathcal{R}_R^-$ and $\mathcal{D}_R$ are laminar and that the sets in $\mathcal{D}_R$ are disjoint by construction. Figure 3.9 illustrates $\mathcal{R}_R^-$ and $\mathcal{D}_R$ for $R = \{1, 2, 3\}$ in the instance and preemptive solution of Example 3.12, see also Figure 3.8.

We claim that, for every $S \in \mathcal{R}_R^-$, there is a set $S' \in \mathcal{D}_R$ such that $S \subseteq S'$. Suppose not, and let $S \in \mathcal{R}_R^-$ be a set so that there is no $S' \in \mathcal{D}_R$ with $S \subseteq S'$. If there is $S' \in \mathcal{D}_R$ with $S \supseteq S'$, then S contains no free elements by definition of $\mathcal{D}_R$. So, S decomposes into subsets in $\mathcal{R}_R^-$ and at least one of these subsets is disjoint from all sets in $\mathcal{D}_R$ (for otherwise $S \subseteq S'$). Let $\overline{S} \in \mathcal{R}_R^-$ with $\overline{S} \subseteq S$ be an inclusion-minimal set that is disjoint from all sets in $\mathcal{D}_R$. Since $\overline{S}$ is inclusion-minimal, it contains free elements. So, it is contained in $\mathcal{D}_R$, which is a contradiction.

We can assume that the sets in $\mathcal{R}_R^-$ (and, therefore, also those in $\mathcal{D}_R$) are covered before time $\pi(R)$ by Algorithm 3 in previous iterations. Otherwise, if ties were broken in favor of R in line 3, we can remove the corresponding sets from $\mathcal{R}_R^-$ and $\mathcal{D}_R$. So, the sets in $\mathcal{R}_R^-$ precede R in the non-preemptive solution, i.e., $\pi(S) < \pi(R)$ for all $S \in \mathcal{R}_R^-$. For the covering time of R, we obtain

$$\begin{aligned}\pi(R) &= \kappa(R) + |\{e \in S \mid S \in \mathcal{R}_R^-,\, \pi(e) < \pi(R)\}| \\ &= \kappa(R) + |\{e \in S \mid S \in \mathcal{D}_R,\, \pi(e) < \pi(R)\}| = \kappa(R) + \sum_{S \in \mathcal{D}_R} \kappa(S).\end{aligned} \tag{3.10}$$

The first equality is by definition of the covering time and since the sets in $\mathcal{R}_R^-$ are precisely those sets that are covered before R and are disjoint from R (i.e., any element in such a set does not contribute to the covering of R). The second equality holds because every $S \in \mathcal{R}_R^-$ is contained in some $S' \in \mathcal{D}_R$ and $\mathcal{D}_R \subseteq \mathcal{R}_R^-$. The last equality is due to the fact that sets in $\mathcal{D}_R$ are disjoint and every set $S \in \mathcal{R}$ is covered as soon as $\kappa(S)$ of its elements have appeared in the linear ordering.

On the other hand, let $\nu_e(t) = \sum_{\tau=1}^{t} x_{e\tau} \in [0;1]$ be the fractional amount of element $e \in U$ that appears before time $t \in [|U|]$ in the preemptive solution. For the covering time of R in the preemptive solution, we have

$$\begin{aligned}\pi'(R) &= \sum_{e \in U} \nu_e(\pi'(R)) = \sum_{e \in R} \nu_e(\pi'(R)) + \sum_{e \notin R} \nu_e(\pi'(R)) \\ &\geq \kappa(R) + \sum_{\substack{e:\, \exists S \in \mathcal{R}_R^- \\ \text{with } e \in S}} \nu_e(\pi'(R)) = \kappa(R) + \sum_{S \in \mathcal{D}_R} \sum_{e \in S} \nu_e(\pi'(R)) \\ &\geq \kappa(R) + \sum_{S \in \mathcal{D}_R} \sum_{e \in S} \nu_e(\pi'(S)) \geq \kappa(R) + \sum_{S \in \mathcal{D}_R} \kappa(S).\end{aligned} \tag{3.11}$$

The first inequality holds since the first sum in the second row contains fewer summands than the last sum in the first row and $\sum_{e \in R} \nu_e(\pi'(R)) \geq \kappa(R)$ by the definition of the covering time, see (3.8). The equality in the second row is similar to (3.10) and the second inequality follows from $\nu_e(t) \geq \nu_e(s)$ for $t \geq s$ and all $e \in U$. Hence, when comparing (3.10) and (3.11), we get $\pi(R) = \kappa(R) + \sum_{S \in \mathcal{D}_R} \kappa(S) \leq \pi'(R)$ for any $R \in \mathcal{R}$ that is considered explicitly by Algorithm 3 after line 5.

Finally, let $S \in \mathcal{R}$ be a set that is never chosen explicitly by the algorithm, but is covered in line 11 because its covering requirement is set to zero in the iteration where $R \in \mathcal{R}$ is considered. Since S was not yet considered by the algorithm in lines 3 or 5, it satisfies $\pi'(S) \geq \pi'(R)$. By construction, $\pi(S) \leq \pi(R)$. With (3.10) and (3.11), we get $\pi(S) \leq \pi(R) \leq \pi'(R) \leq \pi'(S)$, which yields the claim. □

In summary, we obtain a 2-approximation algorithm for generalized min-sum set cover on laminar sets. As before, we get an approximation factor of 2 even for the more general weighted version with weights $w_R \in \mathbb{N}$ for all $R \in \mathcal{R}$.

Corollary 3.14
There is a 2-approximation algorithm for weighted laminar generalized min-sum set cover.

Proof. First, we use the algorithm of [88] to obtain a 2-approximate solution for preemptive laminar generalized min-sum set cover. Note that the algorithm in [88] also works for the weighted variant. Then, we apply Algorithm 3 to transform this preemptive solution into a non-preemptive one. By Theorem 3.13, the objective value of the solution returned by Algorithm 3 is $\sum_{R\in\mathcal{R}} w_R\,\pi(R) \leq \sum_{R\in\mathcal{R}} w_R\,\pi'(R)$. Since preemptive generalized min-sum set cover is a relaxation of GMSSC and the algorithm of [88] is 2-approximate, we have $\sum_{R\in\mathcal{R}} w_R\,\pi'(R) \leq 2\sum_{R\in\mathcal{R}} w_R\,\pi^*(R)$, where $\pi^*(R)$ are the covering times of an optimum solution for the instance of laminar GMSSC. □

For preemptive GMSSC in general, the approximation factor of 2 presented in [88] is essentially best possible assuming a variant of the Unique Games Conjecture [100]. Note that there is no benefit in preempting jobs in AND-scheduling, which can be seen as a special case of GMSSC [166]. Since a factor of 2 is essentially best possible for (preemptive) AND-scheduling under a variant of the Unique Games Conjecture [100, 15], this conditional lower bound translates to preemptive generalized min-sum set cover. Bansal et al. [13] showed that the gap between the costs of an optimal non-preemptive and an optimal preemptive solution for general GMSSC can be as large as 4. Hence, using a preemptive solution to construct a non-preemptive one as proposed by [88] cannot yield an approximation factor strictly better than 8 for GMSSC in general.

Note that it is not clear whether preemptive generalized min-sum set cover (and, thus, the non-preemptive variant) is NP-hard at all for laminar sets. In particular, a polynomial-time algorithm for (weighted) laminar generalized min-sum set cover might be possible.

3.5 A Framework for Approximating Scheduling Problems

We now present a framework to obtain 4η-approximation algorithms for various scheduling and min-sum ordering problems, assuming that we have an η-approximation for finding a density-maximizing set to start with at hand. This framework is a generalization of the histogram proof in [45], the idea of which was also used in similar proofs in, e.g., [159, 90, 122, 81]. To stay consistent with the overall terminology in this thesis, we tailor the framework to scheduling problems only. The general case with more general cost and weight functions, as well as applications in scheduling, pipelined set cover and boolean function evaluation, is studied in [75]. That is, the content of this section is not solely restricted to OR-scheduling, but applies to any sort of single-machine scheduling problem to minimize the sum of weighted completion times.

3.5.1 A Density-Maximizing Greedy Algorithm

Recall that the Greedy algorithm of [18, 44, 45] for min-sum set cover chooses the element that covers the most uncovered sets next. For pipelined set cover, Munagala et al. [128] considered the natural extension of the Greedy algorithm, which chooses the element $e \in U$ that maximizes the ratio $\frac{w(\mathcal{R}_e)}{c_e}$ next, where $\mathcal{R}_e$ are the uncovered sets that contain e. The same idea of choosing (or rather approximating) a feasible subset of elements that maximizes the ratio of covered weight to incurred cost was used to obtain approximation algorithms for precedence-constrained min-sum set cover [122] and expanding search [81].

Suppose we are given an arbitrary scheduling problem on a set of jobs N. Let $\overline{\mathcal{FS}} \subseteq 2^N$ be the equivalent of feasible starting sets (Definition 1.14), i.e., $\overline{\mathcal{FS}}$ contains exactly those subsets of jobs that can start in a feasible schedule. For scheduling with precedence constraints, regardless of whether AND, OR or AND/OR, $\overline{\mathcal{FS}}$ contains the initial sets that are closed under the respective precedence constraints. Recall the definition of the density $\rho(S) = \frac{w(S)}{p(S)}$ of a set $S \subseteq N$. We assume in the following that we are given an η-approximation algorithm with $\eta \geq 1$ for finding a set of maximum density in $\overline{\mathcal{FS}}$. Note that $\eta = 1$ means that we can find such a density-maximizing set in polynomial time. In Section 3.5.2, we present polynomial-time algorithms for finding a density-maximizing feasible starting set for $1 \,|\, \textit{or-prec} = \textit{bipartite} \,|\, \sum w_j C_j$ and $1 \,|\, \textit{or-prec} = \textit{intree} \,|\, \sum w_j C_j$.

The algorithm works as follows and is referred to as *density-maximizing Greedy*. We start with an empty schedule and the set of jobs N. First, we apply the η-approximation algorithm to get a set $S \in \overline{\mathcal{FS}}$ with $\rho(S) \geq \frac{1}{\eta}\rho^*$, where ρ^* is the maximum density of any set in $\overline{\mathcal{FS}}$. Then, we append the jobs in S in any feasible order at the end of the current schedule, and remove the jobs in S from the instance. Finally, we update the set $\overline{\mathcal{FS}} \subseteq 2^{N \setminus S}$ accordingly, and repeat. The technique is similar to (and unifies) results in AND-scheduling [157, 85, 152, 25, 120], min-sum set cover [18, 45, 122] and expanding search [6, 46, 81].

Theorem 3.15
Given an η-approximation algorithm for finding a density-maximizing set in $\overline{\mathcal{FS}}$, the density-maximizing Greedy algorithm is a 4η-approximation algorithm for the initial scheduling problem.

Proof. The proof is fairly similar to the histogram proof in [45]. Suppose the density-maximizing Greedy terminates after h iterations, where we added a η-approximate density-maximizing set to the end of the current schedule in each iteration. For $\ell \in [h]$, let J_ℓ be the set of jobs that is scheduled in the ℓ-th iteration, and let $S_\ell = \bigcup_{q=\ell}^{h} J_q$ be the set of remaining jobs at the beginning of iteration ℓ. The set of subsets of jobs that can be appended to the current schedule at the beginning of iteration ℓ is denoted by $\overline{\mathcal{FS}}_\ell$, i.e., $\overline{\mathcal{FS}}_\ell \subseteq 2^{S_\ell}$ contains the sets $S \subseteq S_\ell$ such that $(N \setminus S_\ell) \cup S \in \overline{\mathcal{FS}}$.

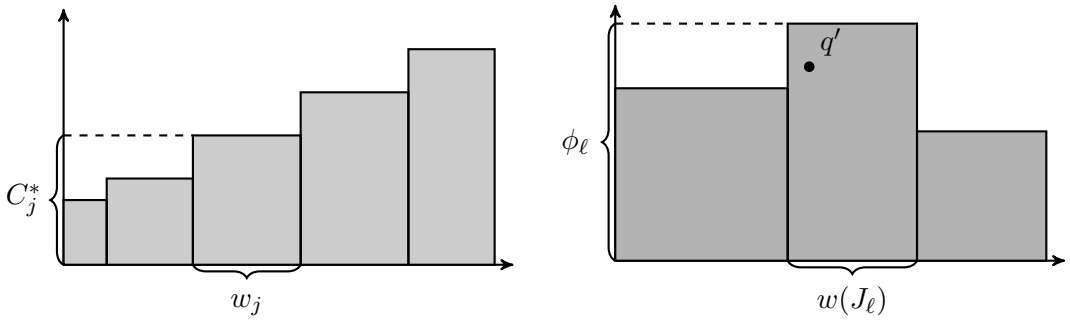

Fig. 3.10: The histograms corresponding to the optimal solution (left) and to the schedule returned by the density-maximizing Greedy (right). The point q' is contained in the ℓ-th column of the second histogram.

By the choice of J_ℓ, we get $\rho(J_\ell) \geq \frac{\rho(J)}{\eta}$ for all $J \in \overline{\mathcal{FS}}_\ell$. The completion time of job $j \in N$ in the schedule returned by the density-maximizing Greedy algorithm and an arbitrary but fixed optimal solution is denoted by C_j^G and C_j^*, respectively.

With $\phi_\ell = \frac{w(S_\ell)}{w(J_\ell)}\, p(J_\ell) = \frac{w(S_\ell)}{\rho(J_\ell)}$ the objective value of the Greedy solution is

$$\begin{aligned} \sum_{j \in N} w_j\, C_j^G &\leq \sum_{\ell=1}^{h} \sum_{j \in J_\ell} w_j \sum_{q=1}^{\ell} p(J_q) = \sum_{\ell=1}^{h} p(J_\ell) \sum_{q=\ell}^{h} w(J_q) \\ &= \sum_{\ell=1}^{h} p(J_\ell)\, w(S_\ell) = \sum_{\ell=1}^{h} w(J_\ell)\, \phi_\ell. \end{aligned} \tag{3.12}$$

We construct two histograms that represent the objective values of the optimal solution and the solution returned by the density-maximizing Greedy, respectively. We show that if we shrink the second histogram by a factor of 2 in horizontal direction and a factor of 2η in vertical direction, it fits into the first one. This then yields the claim.

The first histogram contains a column for each job $j \in N$ with width w_j and height C_j^* in the order the jobs appear in the optimal solution. Note that the height of the columns is non-decreasing and that the total area of the histogram is equal to the optimal objective value, $\sum_{j \in N} w_j C_j^*$, see Figure 3.10 (left). The second histogram consists of h columns, one for each iteration of the density-maximizing Greedy, in the order the iterations appear in the algorithm. The width of column $\ell \in [h]$ is $w(J_\ell)$, and its height is equal to ϕ_ℓ, see Figure 3.10 (right). The total area of the second histogram is equal to $\sum_{\ell=1}^{h} w(J_\ell)\, \phi_\ell \geq \sum_{j \in N} w_j\, C_j^G$, see (3.12).

Now, we shrink the second histogram by a factor 2 in width and a factor 2η in height, and align it to the right, i.e., such that the lower right corner of the rightmost column has coordinates $(w(N), 0)$, see Figure 3.11. The total area of the shrunk histogram is equal to $\frac{1}{4\eta} \sum_{\ell=1}^{h} w(J_\ell)\, \phi_\ell$. We claim that the shrunk histogram is completely contained

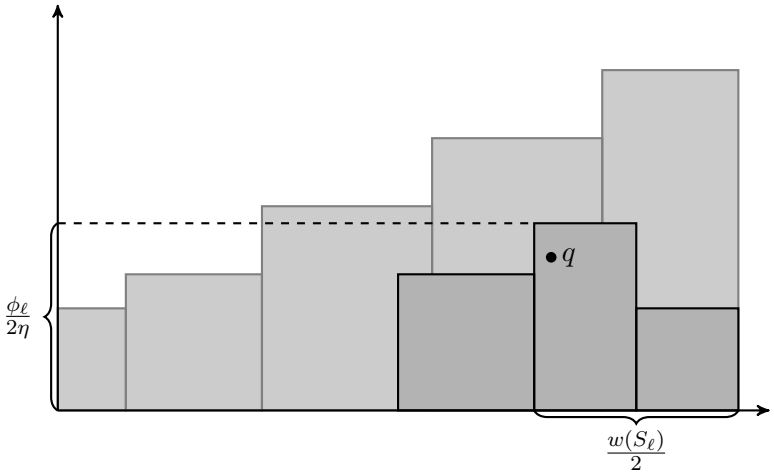

Fig. 3.11: The shrunk histogram (dark gray) aligned right inside the first histogram (light gray). The point q corresponds to q' in Figure 3.10 (right) after shrinking.

in the first histogram. This then implies that the area of the shrunk histogram is less or equal than $\sum_{j\in N} w_j C_j^*$. Together with (3.12) this yields

$$\sum_{j\in N} w_j\, C_j^G \leq \sum_{\ell=1}^{h} w(J_\ell)\,\phi_\ell \leq 4\eta \sum_{j\in N} w_j\, C_j^*. \tag{3.13}$$

To prove the claim, let q' be a point in the second histogram, and suppose it is contained in column ℓ. Let q be the corresponding point in the shrunk histogram. That is, the height of q is at most $\frac{1}{2\eta}\phi_\ell$, and its distance to the right is at most $\frac{1}{2}\sum_{q=\ell}^{h} w(J_q) = \frac{1}{2}w(S_\ell)$.

Let $\rho^* = \max\{\rho(J) \mid J \in \overline{\mathcal{FS}}_\ell\}$, and recall that J_ℓ satisfies $\rho(J_\ell) \geq \frac{1}{\eta}\rho^*$. No schedule (not even an optimal one) can cover more than an amount of $\lambda\rho^*$ of weight of the jobs in S_ℓ during λ time units. Hence, within $\frac{1}{2\eta}\phi_\ell$ time units, the optimal solution cannot cover more than an amount of $\frac{1}{2\eta}\phi_\ell\,\rho^* \leq \frac{1}{2\eta}\phi_\ell\,\eta\,\rho(J_\ell) = \frac{1}{2}\phi_\ell\,\rho(J_\ell)$ of weight of jobs in S_ℓ. So, at time $\frac{1}{2\eta}\phi$, the remaining unscheduled weight of jobs in S_ℓ is at least

$$w(S_\ell) - \frac{1}{2\eta}\phi_\ell\,\rho^* \geq w(S_\ell) - \frac{1}{2\eta}\phi_\ell\,\eta\,\rho(J_\ell) = w(S_\ell) - \frac{1}{2}\frac{w(S_\ell)}{\rho(J_\ell)}\rho(J_\ell) = \frac{1}{2}w(S_\ell). \tag{3.14}$$

Thus, the point $\left(w(N) - \frac{w(S_\ell)}{2}, \frac{\phi_\ell}{2\eta}\right)$ is contained in the first histogram. Since q is to the lower right of $\left(w(N) - \frac{w(S_\ell)}{2}, \frac{\phi_\ell}{2\eta}\right)$ and the column heights of the first histogram are non-decreasing, q is contained in the first histogram. This proves the claim. □

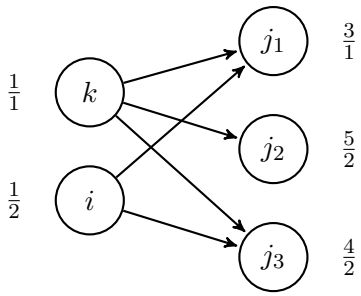

Fig. 3.12: An instance of bipartite OR-scheduling with $A = \{i, k\}$ and $B = \{j_1, j_2, j_3\}$. The ratio next to job j is $\frac{w_j}{p_j}$. The sets are $J_k = \{k, j_1, j_2\}$ and $J_i = \{i, j_1, j_3\}$ with $\rho(J_k) = \frac{9}{4} > \frac{8}{5} = \rho(J_i)$. For the construction of J_k, we have $\rho(\{j_1\}) > 1 = \rho(\{k\})$, $\rho(\{j_2\}) > \frac{4}{2} = \rho(\{k, j_1\})$ and $\rho(\{j_3\}) < \frac{9}{4} = \rho(J_k)$, so j_3 is not included into J_k.

Note that the histogram of [45] in Figure 3.10 (left) is just the flipped two-dimensional Gantt chart of [42, 63]. Successively scheduling a density-maximizing set is similar to Sidney's decomposition [152] and the algorithms of [25, 120] for AND-scheduling. However, we do not get an approximation factor of 2 (or 2η) for the density-maximizing Greedy algorithm via this approach, because, e.g., for scheduling with OR-precedence constraints, there is no Sidney-like decomposition in general. For an example, we refer to the instance of min-sum vertex cover in [45] showing that the Greedy algorithm for min-sum set cover is tight.

3.5.2 Two 4-Approximation Algorithms for OR-Scheduling Problems

In this section, we present polynomial-time algorithms to compute density-maximizing feasible starting sets for OR-scheduling with precedence graphs that are bipartite or in the form of an intree. Thus, the density-maximizing Greedy algorithm described in the previous section is a 4-approximation algorithm for $1 \,|\, \textit{or-prec} = \textit{bipartite} \,|\, \sum w_j C_j$ and $1 \,|\, \textit{or-prec} = \textit{intree} \,|\, \sum w_j C_j$.

Recall that the related AND-scheduling problems $1 \,|\, \textit{prec} = \textit{intree} \,|\, \sum w_j C_j$ and $1 \,|\, \textit{prec} = \textit{outtree} \,|\, \sum w_j C_j$ are solvable in polynomial time [85], whereas $1 \,|\, \textit{prec} = \textit{bipartite} \,|\, \sum w_j C_j$ is NP-hard [166]. Since AND-precedence constraints and OR-precedence constraints coincide on outtrees, we get that $1 \,|\, \textit{or-prec} = \textit{outtree} \,|\, \sum w_j C_j$ can be solved in polynomial time. For technical reasons, we define $\rho(\emptyset) := -1$. So, $\rho(\emptyset) < 0 \leq \rho(S)$ for any non-trivial $S \subseteq N$.

Bipartite OR-Scheduling. Consider an instance of $1 \,|\, \textit{or-prec} = \textit{biparitite} \,|\, \sum w_j C_j$ with precedence graph $G = (A \dot\cup B, E_\vee)$ and set of feasible starting sets $\mathcal{FS} \subseteq 2^{A \dot\cup B}$. W.l.o.g., we can assume that all jobs without predecessors are contained in A, i.e., $j \in B$ implies $\mathcal{P}(j) \neq \emptyset$. Hence, the jobs in A are the singleton feasible starting sets, i.e., $\{i\} \in \mathcal{FS}$ for all $i \in A$.

Input: Instance of $1 \,|\, or\text{-}prec = bipartite \,|\, \sum w_j C_j$
Output: Set $J \in \mathcal{FS}$ with $\rho(J) = \max\{\rho(S) \,|\, S \in \mathcal{FS}\}$

for $i \in A$ **do**
  $J_i \leftarrow \{i\}$, $S_i \leftarrow \{b \in B \,|\, i \in \mathcal{P}(b)\}$ and $q \leftarrow 1$;
  Sort jobs in $S_i = \{b_1^i, \ldots, b_{|S_i|}^i\}$ in non-increasing order of their density;
  while $q \leq |S_i|$ *and* $\rho(\{b_q^i\}) > \rho(J_i)$ **do**
    $J_i \leftarrow J_i \cup \{b_q^i\}$ and $q \leftarrow q + 1$;
  end
end
Let $k \in \operatorname{argmax}\{\rho(J_i) \,|\, i \in A\}$;
return J_k;

Algorithm 4: An algorithm to compute a density-maximizing set in $\mathcal{FS}$ for bipartite OR-scheduling.

We define a set $J_i \in \mathcal{FS}$ for every $i \in A$ so that J_k with $\rho(J_k) = \max\{\rho(J_i) \,|\, i \in A\}$ is a density-maximizing feasible starting set (Lemma 3.16). For $i \in A$, we set $J_i := \{i\} \cup B_i$ where $B_i = \{b_1^i, \ldots, b_\ell^i\} \subseteq \{b \in B \,|\, i \in \mathcal{P}(b)\}$ is inclusion-maximal such that

(*i*) $\rho(\{b_1^i\}) \geq \rho(\{b_2^i\}) \geq \cdots \geq \rho(\{b_\ell^i\}) \geq \rho(\{b\})$ for all $b \in \{j \in B \setminus B_i \,|\, i \in \mathcal{P}(j)\}$, and
(*ii*) $\rho(\{b_q^i\}) > \rho(\{i, b_1^i, \ldots, b_{q-1}^i\})$ for all $q \in [\ell]$.

The sets J_i can be constructed in polynomial time by successively adding a successor of i with highest density to J_i if this increases the overall density of the set, see Algorithm 4. Figure 3.12 depicts an instance and the corresponding sets.

Lemma 3.16
Algorithm 4 computes a density-maximizing feasible starting set for bipartite precedence graphs $G = (A \dot\cup B, E_\vee)$.

Proof. Let $S \in \mathcal{FS}$ be an inclusion-minimal density-maximizing feasible starting set, and let $k \in \operatorname{argmax}\{\rho(J_i) \,|\, i \in A\}$ be the job whose set J_k is returned by Algorithm 4. We show that $\rho(S) = \rho(J_k)$. Note that $A \cap S \neq \emptyset$, since $S \in \mathcal{FS}$ and $\mathcal{P}(b) \neq \emptyset$ for all $b \in B$. Further, $\rho(\{j\}) \leq \rho(S)$ for every $j \notin S$ with $j \in A$ or $j \in B$ with $\mathcal{P}(j) \cap S \neq \emptyset$ because otherwise we could add j to S and obtain a feasible starting set $S \cup \{j\} \in \mathcal{FS}$ with $\rho(S \cup \{j\}) > \rho(S)$. We claim that $J_i \subseteq S$ for all $i \in A \cap S$.

Let $i \in A \cap S$ and suppose by contradiction $J_i \not\subseteq S$. Recall that $J_i = \{i\} \cup B_i$ where $B_i = \{b_1^i, \ldots, b_\ell^i\} \subseteq \{b \in B \,|\, i \in \mathcal{P}(b)\}$ satisfies (*i*) and (*ii*). Let $h \in [\ell]$ be minimal such that $b_h^i \in B_i \setminus S$, i.e., $\{i, b_1^i, \ldots, b_{h-1}^i\} \subseteq S$. By the above observations and the construction of J_i, it holds $\rho(S) \geq \rho(\{b_h^i\}) > \rho(\{i, b_1^i, \ldots, b_{h-1}^i\})$ (condition (*ii*)) and $\rho(S) \geq \rho(\{b_h^i\}) \geq \rho(\{b_q^i\}) \geq \rho(\{b\})$ for all $q \geq h$ and $b \in B \setminus B_i$ with $i \in \mathcal{P}(b)$ (condition (*i*)). But then $S \setminus (\{i\} \cup \{b \in B \,|\, i \in \mathcal{P}(b)\})$ is a feasible starting set with strictly higher density than S. Hence, $J_i \subseteq S$ for all $i \in A \cap S$.

Let $i \in A \cap S$ and suppose $S \supsetneq J_i$, i.e., $S = J_i \dot\cup S'$ for some $S' \neq \emptyset$. Since S was chosen to be inclusion-minimal, $\rho(S) \geq \rho(J_i) \geq \rho(\{b\})$ for all $b \in \{j \in B \setminus B_i \mid i \in \mathcal{P}(j)\}$ implies $S \cap \{j \in B \setminus B_i \mid i \in \mathcal{P}(j)\} = \emptyset$. Hence, $S' \in \mathcal{FS}$ is a feasible starting set as well. The density of S is equal to

$$\rho(S) = \frac{w(S)}{p(S)} = \frac{w(J_i) + w(S')}{p(J_i) + p(S')} = \lambda\rho(J_i) + (1-\lambda)\rho(S'), \tag{3.15}$$

where $\lambda = \frac{p(J_i)}{p(J_i)+p(S')} \in [0;1]$. Maximality of $\rho(S)$ and (3.15) imply $\rho(S) = \rho(J_i) = \rho(S')$. This proves the claim. □

Due to Lemma 3.16, we can apply the density-maximizing Greedy algorithm from Section 3.5.1 and use Algorithm 4 to compute a density-maximizing set in each iteration. After scheduling the set J_k that was returned by Algorithm 4, we have to update the sets A and B. That is, we remove J_k from the instance and move the jobs in $\{j \in B \mid k \in \mathcal{P}(j)\}$ into A (and delete all ingoing arcs), since these jobs are now available. This gives a 4-approximation for bipartite OR-scheduling, which generalizes the results of [45, 128] for min-sum set cover and pipelined set cover. The following theorem follows from Theorem 3.15 and Lemma 3.16.

Theorem 3.17
The density-maximizing Greedy together with Algorithm 4 is a 4-approximation algorithm for $1 \mid or\text{-}prec = bipartite \mid \sum w_j C_j$.

OR-Scheduling on Intrees. We can also compute density-maximizing feasible starting sets in polynomial time for $1 \mid or\text{-}prec = intree \mid \sum w_j C_j$. Let $G = (N, E_\vee)$ be the precedence graph and recall that every job has at most one successor in G if G is an inforest. Recall the definition of a *path*, which is a subset of jobs $\{j_1, \dots, j_\ell\} \subseteq N$ such that $(j_q, j_{q+1}) \in E_\vee$ for all $q \in [\ell - 1]$. A *stem* in G is a path where j_1 has no predecessors, i.e., a stem is a path that is a feasible starting set. Lemma 3.18 shows that a density-maximizing stem in G is also a density-maximizing feasible starting set.

Lemma 3.18
A density-maximizing feasible starting set for precedence graphs $G = (N, E_\vee)$ *in form of an inforest can be computed in polynomial time.*

Proof. Since $G = (N, E_\vee)$ is an inforest, the number of paths starting at a given node is bounded by the total number of nodes $|N| = n$. Therefore, the total number of stems is $\mathcal{O}(n^2)$. So, we can enumerate all stems and pick the one with highest density. It remains to be shown that a density-maximizing stem is indeed also a density-maximizing feasible starting set.

To this end, let $S \in \mathcal{FS}$ be a density-maximizing feasible starting set, and suppose S is not a stem. That is, S contains at least two jobs i and j without a predecessor.

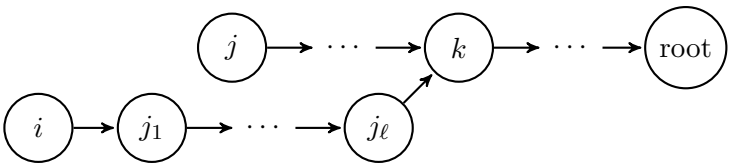

Fig. 3.13: Paths starting at i and j to "their" root in $G[S]$ and their intersection point k.

Note that the induced subgraph $G[S]$ is also an inforest. Since every job has at most one successor, any job without a predecessor induces a unique path to a root of $G[S]$. Let $k \in S$ be the job where the unique paths of i and j in $G[S]$ meet, see Figure 3.13. (Note that k does not exist if i and j are in different connected components.)

Let $J_i = \{i, j_1, \dots, j_\ell\}$ be the set of jobs on the path from i to k such that $j_\ell \in \mathcal{P}(k)$. If k does not exist, let J_i be the path starting at i to the corresponding root in $G[S]$. By construction, $J_i \in \mathcal{FS}$ is a stem and $S' = S \setminus J_i \in \mathcal{FS}$ is a feasible starting set. Similar to (3.15) in the proof of Lemma 3.16, we obtain $\rho(S) = \rho(J_i) = \rho(S')$ by the maximality of $\rho(S)$. This proves the claim. □

Observe that, if we remove a stem from an inforest, the digraph decomposes into intrees again. That is, we can successively remove a density-maximizing stem from the precedence graph and apply the density-maximizing Greedy algorithm from Section 3.5.1. By Lemma 3.18, this gives a 4-approximation algorithm for OR-scheduling on intrees.

Theorem 3.19
The density-maximizing Greedy is a 4-approximation algorithm for $1 \,|\, or\text{-}prec = intree \,|\, \sum w_j C_j$ *if we schedule a density-maximizing stem in each iteration.*

Note that the proof of Lemma 3.18 also works for more general digraphs such as *multitrees*, which were proposed by [48] as a generalization of rooted trees. A directed acyclic graph $G = (N, E)$ is called a *multitree* if, for each $j \in N$, the set of nodes that are reachable from j form an outtree. Equivalently, G is a multitree if it is *diamond free*, i.e., for every pair of nodes $i, j \in N$, there is at most one path from i to j [48].

Similar to Lemma 3.18, a density-maximizing outtree rooted at a job without predecessors is a density-maximizing feasible starting set if the graph is a multitree. One can easily verify that the proof of Lemma 3.18 still works if we informally replace *stem* and *path* by *outtree* and *inforest* by *multitree*. In particular, if $G[S]$ is not an outtree, where S is a density-maximizing set, then there are two jobs $i, j \in S$ without a predecessor. In this case, i and j induce unique outtrees in $G[S]$, instead of paths to a root as in the proof of Lemma 3.18. (Note that there might exist several jobs like k where the two outtrees "meet".) Let $S_i, S_j \subseteq S$ be the set of jobs that are contained in the outtree in $G[S]$ rooted at i and j, respectively. For $J_i = S_i \setminus S_j$, we obtain that $\rho(J_i) = \rho(S \setminus J_i) = \rho(S)$ similar to (3.15).

Let $i \in N$ with $\mathcal{P}(i) = \emptyset$. Since the nodes reachable from i form an outtree in G by definition of a multitree, we can use a dynamic program similar to the subalgorithm of Horn [85] to compute a density-maximizing outtree J_i rooted at i. Then, we choose the outtree with maximum density over all jobs without predecessors. Note that the precedence graph is still diamond free after removing an outtree J_i (and all arcs going into J_i). Hence, the density-maximizing Greedy is also 4-approximate in this more general setting.

3.6 Open Problems

Bipartite OR-scheduling is not just NP-hard as a generalization of min-sum set cover, but already for the simplest non-trivial processing times and weights, see Theorem 3.6. On the positive side, the extension of the Greedy algorithm for min-sum set cover, namely the density-maximizing Greedy described in Section 3.5, is 4-approximate for bipartite OR-scheduling in general (Theorem 3.17). Feige, Lovász and Tetali [45] showed that no polynomial-time algorithm can do better than 4, even for min-sum set cover. However, for unit processing times or unit weights, bipartite OR-scheduling might admit better approximation guarantees.

Problem 3.20
Are there $(4-\varepsilon)$-approximation algorithms for $1 \,|\, \textit{or-prec} = \textit{bipartite},\ p_j = 1 \,|\, \sum w_j C_j$ or $1 \,|\, \textit{or-prec} = \textit{bipartite} \,|\, \sum C_j$ for a fix $\varepsilon > 0$?

In Section 3.4, we observed that variants of GMSSC are solvable in polynomial time if the collection of subsets is laminar. The proof of Lemma 3.7 fails for laminar pipelined set cover, since the covering cost of a set that is covered by an element in S changes if we swap e and f (in the notation of the proof of Lemma 3.7). The procedure of transforming an instance of laminar GMSSC to an equivalent instance of MLSC (Algorithm 2) does not work if the requirements are not of an "all-but-constant" form. Further, it is not clear whether laminar generalized min-sum set cover, for which we propose a 2-approximation algorithm (Corollary 3.14), is NP-hard at all.

Problem 3.21
Is there a polynomial-time algorithm for laminar generalized min-sum set cover?

All results in Section 3.4 use the property that the collection of subsets $\mathcal{R}$ is laminar. Since all-but-K min-sum set cover generalizes min-sum set cover (by choosing $K \in \mathbb{N}$ sufficiently large), there is no $(4-\varepsilon)$-approximation algorithm for all-but-K min-sum set cover with arbitrary $\mathcal{R}$, unless P = NP [45]. Note that MLSC is a special case of $1 \,|\, \textit{prec} \,|\, \sum w_j C_j$ and can, therefore, be approximated within a factor of 2 [148, 71, 30, 25, 120]. In Section 4.3.2, we propose a 4-approximation algorithm for the special case where $K = 1$. It seems natural to ask whether an approximation factor of 4 is possible for arbitrary $K \in \mathbb{N}$.

Problem 3.22
Is there a 4-approximation algorithm for all-but-K min-sum set cover for all $K \in \mathbb{N}$?

The best-known approximation guarantee for generalized min-sum set cover (with arbitrary requirements) so far is 4.642 [13]. The authors in [88, 13] conjecture that GMSSC admits a 4-approximation algorithm as does min-sum set cover. Note that generalized min-sum set cover falls into the framework discussed in Section 3.5. That is, an η-approximation algorithm with $\eta \leq 1.16$ or a polynomial-time algorithm for finding a density-maximizing subset of elements and covered sets would improve the approximation factor for GMSSC.

Problem 3.23
Is there a 4-approximation algorithm for generalized min-sum set cover?

It would be of interest to find further applications of the density-maximizing Greedy framework in the variant discussed in this thesis or in the more general setting of [75]. In particular, for applications that do not seem to be as obvious as scheduling problems, this framework might offer new possibilities to obtain approximation algorithms. Also other special cases of OR-scheduling fall into this Greedy framework and might admit 4η-approximation algorithms like bipartite OR-scheduling (Theorem 3.17) or OR-scheduling on intrees (Theorem 3.19). Note that it is not clear whether the latter problem is NP-hard or solvable in polynomial time.

Problem 3.24
Is there a polynomial-time algorithm for $1 \,|\, \textit{or-prec} = \textit{intree} \,|\, \sum w_j C_j$?

Recall that OR-scheduling is strongly NP-hard, even if the precedence graph is bipartite (Theorem 3.6) and that there are constant-factor approximation algorithms for some special cases (Theorems 3.17 and 3.19). Since the only known lower bound on the approximability of OR-scheduling is 4 from min-sum set cover [45], it would be interesting to show whether there is a constant-factor approximation algorithm for OR-scheduling in general.

Problem 3.25
Is there a constant-factor approximation for $1 \,|\, \textit{or-prec} \,|\, \sum w_j C_j$?

Chapter 4

Linear Programming Relaxations and LP Based Algorithms

In this chapter, we study various LP relaxations, and present LP based approximation algorithms for scheduling problems that generalize precedence-constrained min-sum set cover and all-but-one min-sum set cover. The results in this chapter were presented at WAOA 2019 [78]. A journal version of this work is available online in [77].

4.1 Related Work and Our Results

Besides discussing standard linear programming relaxations for scheduling problems in the context of OR-precedence constraints, we present new approximation algorithms for $1 \,|\, ao\text{-}prec = A \dot{\vee} B \,|\, \sum w_j C_j$ (Definition 1.17) and all-but-one min-sum set cover (Definition 3.3) in this chapter. Recall the min-sum covering problems of Sections 1.2.1 and 3.2.1 and their connection to scheduling, see Figures 1.5 and 3.2.

In Section 4.2, we discuss three classical LP formulations for AND-scheduling from the literature. We present new approximation algorithms that are based on time-indexed LP formulations for bipartite AND/OR-scheduling and the precedence-constrained variant of min-sum vertex cover in Section 4.3.1. Further, we give a 4-approximation algorithm for a scheduling problem that generalizes all-but-one min-sum set cover in Section 4.3.2. In Sections 4.4 and 4.5, we study other standard LP formulations for scheduling problems in linear ordering and completion time variables, respectively. We present classes of valid inequalities for each of these formulations and show that the integrality gaps grow linear in the number of jobs if OR-precedence constraints are involved, even with the additional constraints. Figure 4.1 gives an overview of the problems that are discussed in Section 4.3 and the related problems in the literature. To avoid redundancy, we do not repeat the relevant results in the literature that were already discussed in Chapter 3 and refer the reader to Section 3.1 instead.

Precedence-Constrained MSSC and Set Cover. As an extension of min-sum set cover, which was studied in [44, 45], Munagala et al. [128] introduced pipelined set cover (Definition 3.2) and showed that the natural extension of the Greedy algorithm is also 4-approximate in this more general setting. Munagala et al. [128] posed as an open problem whether pipelined set cover still admits a constant-factor approximation if AND-precedence constraints in form of a partial order $\prec$ are incorporated on the elements. That is, any feasible linear ordering $\pi : U \to [|U|]$ must satisfy $\pi(e) < \pi(e')$ whenever $e \prec e'$. This question was partly settled by McClintock, Mestre and Wirth [122]. They presented a $4\sqrt{|U|}$-approximation algorithm for precedence-constrained min-sum set cover (Definition 1.11). The algorithm uses a $\sqrt{|U|}$-approximative Greedy algorithm on a problem called *max-density precedence-closed subfamily* together with a histogram argument similar to [45] and Section 3.5.

McClintock, Mestre and Wirth [122] also proposed a reduction from the so-called *planted dense subgraph conjecture* [24] to precedence-constrained MSSC. Roughly speaking, the conjecture says that, for all $\varepsilon > 0$, there is no polynomial-time algorithm that can decide with advantage $> \varepsilon$ whether a random graph on m vertices is drawn from $(m, m^{\alpha-1})$ or contains a subgraph drawn from $(\sqrt{m}, \sqrt{m}^{\beta-1})$ for certain parameters $0 < \alpha, \beta < 1$.[21] If the conjecture holds true, then this implies that there is no $\mathcal{O}(|U|^{1/12-\varepsilon})$-approximation algorithm for precedence-constrained MSSC [122].

In Section 4.3.1, we propose a $(2\Delta + \varepsilon)$-approximation algorithm for bipartite AND/OR-scheduling, $1 \,|\, ao\text{-}prec = A \dot{\vee} B \,|\, \sum w_j C_j$, where $\Delta = \max\{|\mathcal{P}(b)| \,|\, b \in B\}$ is the maximum number of OR-predecessors of any one job in B. Recall that bipartite AND/OR-scheduling generalizes precedence-constrained MSSC, see Section 1.2.2 and Figure 1.5. This implies a 2Δ-approximation for precedence-constrained min-sum set cover and a 4-approximation for precedence-constrained min-sum vertex cover. We further argue how the reduction from the planted dense subgraph problem in [122] implies that no $\mathcal{O}(\Delta^{1/3-\varepsilon})$-approximation algorithm exists for bipartite AND/OR-scheduling, if the planted dense subgraph conjecture holds true.

The ordinary Set Cover and the equivalent Hitting Set problem [98] are also special cases of $1 \,|\, ao\text{-}prec = A \dot{\vee} B \,|\, \sum w_j C_j$.[22] Consider an instance of the Hitting Set problem with a finite set of elements U and a family of sets $\mathcal{R} \subseteq 2^U$. We can introduce a job in A with $p_a = 1$ and $w_a = 0$ for every element in U and a job in B with $p_b = w_b = 0$ for every set in $\mathcal{R}$. The OR-precedence constraints $E_\vee \subseteq A \times B$ are given by the covering graph of the instance. Further, we include an additional job x in B with $p_x = 0$ and $w_x = 1$, and introduce an arc $(b, x) \in E_\wedge$ for every job $b \in B \setminus \{x\}$.

[21] A random (undirected) graph drawn from (m, p) contains m vertices and the probability of the existence of an edge between any two vertices is equal to p.

[22] For Set Cover, one tries to find a subset of $\mathcal{R}$ of minimum cardinality that covers all elements in U and, for Hitting Set, one wants to determine a subset $U' \subseteq U$ of minimum cardinality such that each set in $\mathcal{R}$ contains an element of U'. Such a subset U' is called a *hitting set*. By exchanging the role of U and $\mathcal{R}$ and reversing all arcs in the covering graph, it becomes clear that Set Cover and Hitting Set are equivalent. Since Hitting Set ("find a subset of elements") is closer to our definition of MSSC ("order the elements"), we use the hitting set terminology in the following.

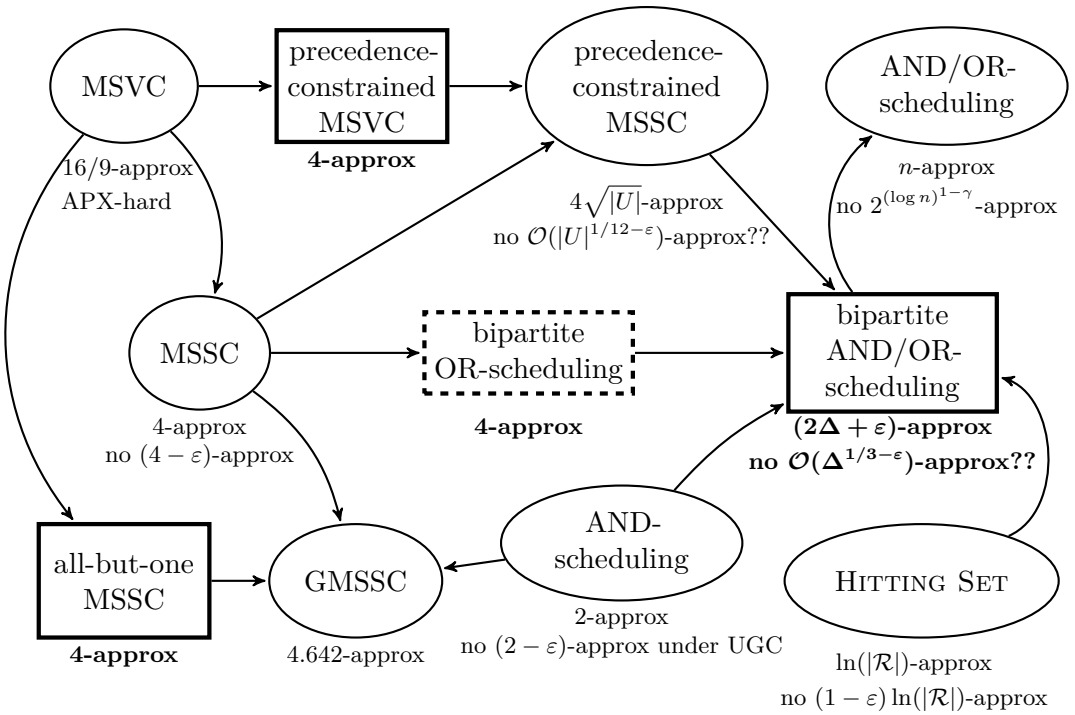

Fig. 4.1: Overview of related problems and results. An arrow from problem Π_1 to Π_2 indicates that Π_2 generalizes Π_1. Problems in rectangular frames are explicitly considered in this thesis, and our results are depicted in bold. Bipartite OR-scheduling (rectangular dashed frame) is discussed in Chapter 3. Lower bounds indicated with "??" are assuming hardness of the planted dense subgraph problem [24].

If the Hitting Set instance admits a hitting set of cardinality ℓ, we first schedule the corresponding ℓ element-jobs in A, so all set-jobs are available for processing at time ℓ. Then, job x can complete at time ℓ, which gives an overall objective value of ℓ. Similarly, any schedule with objective value equal to ℓ implies that all set-jobs are completed before time ℓ, so there exists a hitting set of size at most ℓ.

Hitting Set can be approximated within a factor of $\ln(|\mathcal{R}|)$ [94, 119, 31], and this is best possible, unless P = NP [39]. The algorithms in [94, 119, 31] are Greedy heuristics. In the unweighted version [94, 119], the element that "hits" most uncovered sets is chosen in each iteration. Chvátal [31] generalized this to the weighted version, where each set and each element is associated with a non-negative weight and cost, respectively. In each step, Greedy chooses the element $e \in U$ that maximizes the ratio of total weight of uncovered sets that contain e to the cost of e [31]. Figure 4.1 illustrates the connections between the problems discussed in this chapter and the relevant literature.

Bar-Yehuda and Even [21] and Hochbaum [82] presented $\max_{R\in\mathcal{R}}|R|$-approximations for Hitting Set. The approximation factor is equal to the maximum cardinality of a set in $\mathcal{R}$. Note that the approximation factor of $\ln(|\mathcal{R}|)$ in [94, 119, 31] is better than $\max_{R\in\mathcal{R}}|R|$ in [21, 82] in general. However, for Vertex Cover (see Section 2.3.2), which is a special case of Set Cover, the algorithms in [21, 82] yield a constant approximation ratio of 2. The algorithm of Bar-Yehuda and Even [21] is purely combinatorial, whereas the algorithm of Hochbaum [82] requires solving the canonical LP for Set Cover. In the reduction from Hitting Set to $1\,|\,ao\text{-}prec = A\dot{\vee}B\,|\,\sum w_jC_j$ described above, $\max_{R\in\mathcal{R}}|R|$ equals the parameter $\Delta = \max\{|\mathcal{P}(b)|\,|\,b\in B\}$ in the approximation factor we present in Section 4.3.1.

Scheduling Polyhedra and AND/OR-Scheduling. We already discussed some related work on AND-scheduling in Section 3.1. The first constant-factor approximation algorithm for $1\,|\,prec|\,\sum w_jC_j$ was proposed by Hall, Shmoys and Wein [73]. It had an approximation factor of $4+\varepsilon$. Their algorithm is based on a time-indexed linear program and α-point scheduling with a fixed value of α.[23] Our algorithms in Section 4.3 are also based on time-indexed linear programs and *random* α-point scheduling, similar to, e.g., [59, 73, 71, 149, 26, 62]. One new element here is to not use a global value for α, but to use different values of α for the jobs in A and B, respectively. This is crucial in order to obtain feasible schedules in the end.

First polyhedral studies for scheduling problems date back to Balas [11] and Wolsey [167]. There are three standard LP formulations in the single-machine scheduling literature, see, e.g., [140] for a survey. The first type of polytopes is based on *time-indexed variables* (introduced by [41, 158]) and forms the basis of the approximation algorithms presented in Section 4.3. Other typical linear programming formulations are based on *linear ordering variables* (introduced by [137]) and *completion time variables* (introduced by [167, 139]). Schulz [148] used an LP in completion time variables to obtain the first 2-approximation algorithm for $1\,|\,prec|\,\sum w_jC_j$. The formulation in linear ordering variables played an important role in better understanding Sidney's decomposition [152, 34], and in uncovering the connection between AND-scheduling and Vertex Cover [30, 34, 7]. We elaborate on these formulations and further related results in Section 4.2.

Scheduling with AND/OR-precedence constraints (Definition 1.16) to minimize the maximum lateness was first considered in [58, 57]. Erlebach, Kääb and Möhring [43] studied the sum of weighted completion times objective on a single machine. They showed that $1\,|\,ao\text{-}prec|\,\sum w_jC_j$ does not admit constant-factor approximations, unless P = NP, which is in contrast to scheduling with AND-precedence constraints only. The inapproximability proof uses a problem called Label Cover [9]. Goldwasser and Motwani [64] reduced Label Cover to an AND/OR-scheduling model where some jobs may be skipped, i.e., do not have to be processed. Erlebach, Kääb and Möhring [43] extended the reduction in [64] to $1\,|\,ao\text{-}prec,\,p_j=1\,|\,\sum w_jC_j$, see also [96].

[23] The concept of α-points is introduced in Section 4.2.3.

Let $0 < c < \frac{1}{2}$ and $\gamma = (\log\log n)^{-c}$. It is NP-hard to approximate the sum of weighted completion times of unit processing time jobs on a single machine within a factor of $2^{(\log n)^{1-\gamma}}$ if AND/OR-precedence constraints are involved [64, 43]. The precise factor comes from the hardness of Label Cover, see [38]. The precedence graph in the reduction of [64, 43] consists of four layers with an OR/AND/OR/AND-structure.

Erlebach, Kääb and Möhring [43] also showed that scheduling the jobs in order of non-decreasing processing times (among the available jobs) yields an n-approximation algorithm for $1 \,|\, ao\text{-}prec \,|\, \sum w_j C_j$. For unit weights, this algorithm, which is commonly known as *shortest processing time first*, has an approximation guarantee of $\sqrt{n}$ [43]. It can easily be verified that bipartite AND/OR-scheduling (Definition 1.17), which we consider in Section 4.3.1, is a special case of the problem considered in [43].

4.2 Preliminaries: LP Formulations for AND-Scheduling

In this section, we present the three classical linear programming formulations for single-machine scheduling problems with AND-precedence constraints. The different types of variables discussed are completion time variables (Section 4.2.1), linear ordering variables (Section 4.2.2) and time-indexed variables (Section 4.2.3), respectively. For a more detailed survey, we refer to, e.g., [140, 114].

4.2.1 Completion Time Variables

The LP relaxation in completion time variables contains one variable C_j for every job $j \in N$, which indicates the completion time of this job. This type of variables was introduced by Wolsey [167] and Queyranne [139], who showed that the completion times of any feasible single-machine schedule satisfy the so-called *parallel inequalities*.

Proposition 4.1 (Wolsey [167], Queyranne [139])
Let N be a set of jobs. The convex hull of all feasible completion time vectors for $1 \,||\, \sum w_j C_j$ is exactly the set of vectors $C \in \mathbb{R}^N$ that satisfy

$$\sum_{j \in S} p_j C_j \geq \frac{1}{2}\Big(\sum_{j \in S} p_j\Big)^2 + \frac{1}{2}\Big(\sum_{j \in S} p_j^2\Big) \qquad \forall\, S \subseteq N. \tag{4.1}$$

We do not prove Proposition 4.1, but we briefly argue why any feasible completion time vector for $1 \,||\, \sum w_j C_j$ satisfies (4.1) using the two-dimensional Gantt charts of [42] and Smith's rule ([157], Proposition 3.4), see also [114]. To shorten notation, we let $g : 2^N \to \mathbb{N}_0$ be defined as $g(S) := \frac{1}{2}\left(\sum_{j \in S} p_j\right)^2 + \frac{1}{2}\left(\sum_{j \in S} p_j^2\right)$. That is, the *parallel inequalities* (4.1) can be written as

$$\sum_{j \in S} p_j C_j \geq g(S) \qquad \text{for all } S \subseteq N. \tag{4.2}$$

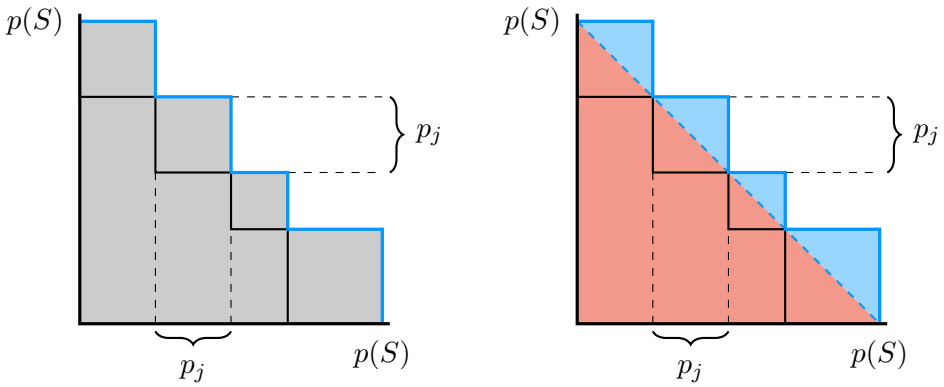

Fig. 4.2: Geometric interpretation of the parallel inequalities (4.1). The gray area underneath the blue line (left) equals the sum of weighted completion times of an optimal schedule with weights $w_j = p_j$ for $j \in S$ and $w_j = 0$ for $j \notin S$. This area can be decomposed into the red and blue areas on the right. The red area equals $\frac{1}{2}\left(\sum_{j\in S} p_j\right)^2$, and the sum of the blue areas is equal to $\frac{1}{2}\sum_{j\in S} p_j^2$. So, the gray area equals $g(S)$.

Consider a subset of jobs $S \subseteq N$ and define weights $w_j = p_j$ for $j \in S$ and $w_j = 0$ for $j \notin S$. Clearly, a schedule that minimizes the sum of weighted completion times with respect to these weights processes all jobs with non-zero weight before all jobs with zero weight (recall that there are no precedence constraints). That is, jobs in S precede those not in S in any optimal schedule. We interpret such a schedule via the corresponding two-dimensional Gantt chart of [42], see Figure 4.2 (left). Observe that the height and width of the rectangle for any $j \in S$ is equal by the choice of the weights. That is, the rectangles corresponding to the jobs in S are squares.

By Smith's rule (Proposition 3.4), any schedule that processes the jobs in S before those not in S is optimal with respect to the objective function $\sum_{j\in N} w_j C_j$, independent of the order of the jobs within S. The area underneath the upper envelope of the rectangles, which equals the objective value, is equal to $g(S)$, see Figure 4.2 (right). So, an optimum solution has objective value equal to $\sum_{j\in N} w_j\, C_j = \sum_{j\in S} p_j\, C_j = g(S)$. Since an optimal solution minimizes the sum of weighted completion time, we obtain that any feasible schedule satisfies $\sum_{j\in S} p_j\, C_j \geq g(S)$.

Note that the parallel inequalities are valid for any arbitrary schedule and that Proposition 4.1 does not consider AND-precedence constraints. We can model AND-constraints in an intuitive way with completion time variables via a constraint

$$C_j \geq C_i + p_j \qquad \text{for all } (i,j) \in E_\wedge. \tag{4.3}$$

Together with the objective function to minimize the sum of weighted completion times, we obtain the following valid LP relaxation for $1\,|\,prec\,|\,\sum w_j C_j$:

$$\min \quad \sum_{j \in N} w_j C_j \tag{4.4a}$$

$$\text{s.t.} \quad \sum_{j \in S} p_j C_j \geq \frac{1}{2}\Big(\sum_{j \in S} p_j\Big)^2 + \frac{1}{2}\Big(\sum_{j \in S} p_j^2\Big) \qquad \forall\, S \subseteq N, \tag{4.4b}$$

$$C_j - C_i \geq p_j \qquad \forall\, (i,j) \in E_\wedge, \tag{4.4c}$$

$$C_j \geq 0 \qquad \forall\, j \in N. \tag{4.4d}$$

The number of constraints of LP (4.4) is exponential. The right hand side $g(S)$ of constraints (4.4b) is a *supermodular* function [139]. Hence, $\sum_{j \in S} p_j C_j - g(S)$ is a *submodular* function in S for fixed values of C_j.[24] A violated inequality can be found by minimizing this submodular function and checking whether its minimum is less than zero, which can be done in (strongly) polynomial time [146, 89].

Queyranne [139] proposed a more efficient separation algorithm for constraints (4.4b): Given a completion time vector, sort the jobs in non-decreasing order of the C_j values, i.e., assume that the jobs are enumerated so that $C_1 \leq C_2 \leq \cdots \leq C_n$. Then, a "most violated" constraint of type (4.4b) is obtained for $S = [\ell]$ for some $\ell \in [n]$, if a violated constraint exists [139]. The running time of this algorithm is $\mathcal{O}(n \log(n))$ for sorting the jobs [124, 139]. Hence, LP (4.4) can be solved in polynomial time [70, 139].

Note that LP (4.4) is just a relaxation for AND-scheduling. That is, any completion time vector that corresponds to a feasible schedule for $1\,|\,prec\,|\,\sum w_j C_j$ is feasible for LP (4.4), but not every feasible vector for the LP can be expressed as a convex combination of completion time vectors of feasible schedules. Queyranne and Wang [142] studied the convex hull of all feasible completion time vectors for $1\,|\,prec\,|\,\sum w_j C_j$. They identified facet-defining inequalities and characterized the convex hull of all feasible completion time vectors if the precedence graph is series-parallel.

Observe that, since $1\,|\,prec\,|\,\sum w_j C_j$ is NP-hard [108, 112], one cannot hope to find a complete description of the convex hull of all feasible completion time vectors similar to Proposition 4.1, unless P = NP. Schulz [148] proposed a 2-approximation algorithm for $1\,|\,prec\,|\,\sum w_j C_j$ that solves LP (4.4) and schedules the jobs in non-decreasing order of their LP values. Hall et al. [71] presented an instance where the optimal objective value of LP (4.4) and the objective value of an optimal feasible schedule are apart by a factor of 2. This implies that the "integrality gap" between an optimal solution of LP (4.4) and an optimal feasible schedule is equal to 2. In Section 4.5, we generalize the parallel inequalities (4.1) to OR-precedence constraints using *generalized minimal chains* (Definition 4.21), and show that a corresponding LP relaxation similar to LP (4.4) exhibits an unbounded gap between optimal LP solution and optimal feasible schedule.

[24] Submodularity and supermodularity are defined in Section 4.5.1.

4.2.2 Linear Ordering Variables

The linear ordering relaxation for single-machine scheduling problems was proposed by Potts [137]. It is based on binary linear ordering variables δ_{ij} that indicate whether job i precedes job j ($\delta_{ij} = 1$) or not ($\delta_{ij} = 0$). To obtain a feasible single-machine schedule, the jobs need to satisfy *total ordering constraints*, i.e., for every pair of distinct jobs, one job must precede the other:

$$\delta_{ij} + \delta_{ji} = 1 \qquad \text{for all } i, j \in N : i \neq j. \tag{4.5}$$

Moreover, the linear ordering needs to be consistent, i.e., if job i precedes j and j precedes k, then i also precedes k. This is achieved by *transitivity constraints*:

$$\delta_{ij} + \delta_{jk} + \delta_{ki} \geq 1 \qquad \text{for all } i, j, k \in N. \tag{4.6}$$

The completion time of job j is the sum of the processing times of all jobs that precede j plus its own processing time. If we set $\delta_{ii} = 1$ for all $i \in N$, the sum of weighted completion times can be written as

$$\sum_{j \in N} w_j C_j = \sum_{j \in N} w_j \sum_{i \in N} p_i \, \delta_{ij} = \sum_{j \in N} \sum_{i \in N} w_j \, p_i \, \delta_{ij}. \tag{4.7}$$

Together with AND-precedence constraints, which can be easily modeled by setting $\delta_{ij} = 1$ for all $(i, j) \in E_\wedge$, we obtain a polynomial size integer program for $1 \,|\, prec \,|\, \sum w_j C_j$. The LP relaxation is obtained by relaxing the integrality constraints $\delta_{ij} \in \{0, 1\}$ to $\delta_{ij} \geq 0$:

$$\begin{aligned}
\min \quad & \sum_{j \in N} \sum_{i \in N} w_j \, p_i \, \delta_{ij} && && \text{(4.8a)} \\
\text{s.t.} \quad & \delta_{ij} + \delta_{ji} = 1 && \forall\, i, j \in N : i \neq j, && \text{(4.8b)} \\
& \delta_{ij} + \delta_{jk} + \delta_{ki} \geq 1 && \forall\, i, j, k \in N, && \text{(4.8c)} \\
& \delta_{ij} = 1 && \forall\, (i, j) \in E_\wedge \text{ or } i = j, && \text{(4.8d)} \\
& \delta_{ij} \geq 0 && \forall\, i, j \in N. && \text{(4.8e)}
\end{aligned}$$

Note that there is a one-to-one correspondence between feasible integer solutions for LP (4.8) and feasible single-machine schedules (without idle time). Schulz [148] and Chudak and Hochbaum [30] used LP (4.8) to derive 2-approximation algorithms for $1 \,|\, prec \,|\, \sum w_j C_j$. Chudak and Hochbaum [30] replaced the transitivity constraints (4.8c) by the valid constraints $\delta_{ki} \leq \delta_{kj}$ for all $(i, j) \in E_\wedge$ and all $k \in N \setminus \{i, j\}$. The resulting linear program, which we denote by CH-LP in the following, can be solved combinatorially by minimum cut computations in a suitable graph [30]. Chudak and Hochbaum [30] observed that $\overline{C}_j = \sum_{i \in N} p_i \, \overline{\delta}_{ij}$, where $\overline{\delta}$ is the optimal solution to CH-LP, is feasible for the completion time LP (4.4). Thus, similar to [148], scheduling the jobs in non-decreasing order of $\overline{C}_j$ values yields a 2-approximate solution.

A similar observation was previously made by Queyranne and Schulz [140] and Schulz [148], who showed that any feasible solution for LP (4.8) defines a feasible solution for the completion time LP (4.4) by setting $C_j = \sum_{i \in N} p_i \, \delta_{ij}$. Chekuri and Motwani [25] gave a lower bound of 2 on the integrality gap of LP (4.8). Together with the results in [140, 148, 30], this implies that the integrality gap of LP (4.8) is equal to 2. In Section 4.4, we present facet-defining inequalities for the linear ordering formulation if OR-precedence constraints are involved, and show that the integrality gap is unbounded, even if we add these additional constraints.

The work of [30] triggered an interesting connection between AND-scheduling and Vertex Cover. Correa and Schulz [34] proposed a new linear relaxation, and showed that solving their LP is, in fact, equivalent to solving CH-LP. The linear program in [34] can be interpreted as an LP relaxation of a Vertex Cover instance on a certain graph [34]. Correa and Schulz [34] also proved that all known 2-approximation algorithms for $1 \,|\, prec \,|\, \sum w_j C_j$ [148, 71, 30, 25, 120] are consistent with Sidney's decomposition [152], even those algorithms [148, 71, 30] that do not use Sidney's result explicitly. Ambühl and Mastrolilli [7] presented a polynomial-time procedure to transform any feasible (integer) solution for CH-LP to a feasible (integer) solution for Potts' original LP (4.8) [137] of the same objective value. Thereby, [34, 7] established that AND-scheduling is, in fact, a Vertex Cover problem. As a consequence, $1 \,|\, prec \,|\, \sum w_j C_j$ can be solved in polynomial time if the precedence graph represents a two-dimensional partial order, which generalizes the result of Lawler [108] for series-parallel graphs. In a follow-up paper, Ambühl et al. [8] used dimension theory of partial orders to derive several new approximation guarantees for AND-scheduling with special precedence graphs.

4.2.3 Time-Indexed Variables

The integrality gaps of the OR-scheduling counterparts of LPs (4.8) and (4.4) grow linear in the number of jobs, see Sections 4.4 and 4.5, respectively. Therefore, we focus on time-indexed LP relaxations for our approximation algorithms in Section 4.3. These formulations are based on binary variables x_{jt} for every job $j \in N$ and every point in time $t \in [T]$, where T is a suitably chosen time horizon. The variable x_{jt} indicates whether job j completes at time t ($x_{jt} = 1$) or not ($x_{jt} = 0$). This formulation was proposed by Dyer and Wolsey [41] and studied by Sousa and Wolsey [158].

Note that a job with processing time p_j cannot complete at any point in time $\{1, \ldots, p_j - 1\}$. This can be modeled by defining a variable x_{jt} only for $t \geq p_j$. As a time horizon, we can choose $T = p(N)$. To obtain a feasible single-machine schedule, every job has to be completed at some point in time, i.e.,

$$\sum_{t=1}^{T} x_{jt} = 1 \qquad \text{for all } j \in N. \tag{4.9}$$

Further, no two jobs may overlap, which is ensured by the following constraint:

$$\sum_{j \in N} \sum_{\tau=\max\{1,t-p_j+1\}}^{t} x_{j\tau} \leq 1 \qquad \text{for all } t \in [T]. \tag{4.10}$$

Intuitively, constraint (4.10) "blocks" the preceding p_j time slots if job j completes at time t. Finally, we can incorporate AND-precedence constraints as in [73] via the constraints

$$\sum_{\tau=1}^{t+p_j} x_{j\tau} \leq \sum_{\tau=1}^{t} x_{i\tau} \qquad \text{for all } (i,j) \in E_\wedge \text{ and } t \in [T - p_j]. \tag{4.11}$$

That is, if j completes at time $t + p_j$, then job i must be completed by time t.

Thus, we obtain the following valid LP relaxation for $1 \,|\, prec \,|\, \sum w_j C_j$ by relaxing the integrality constraints $x_{jt} \in \{0,1\}$ to $x_{jt} \geq 0$:

$$\min \qquad \sum_{j \in N} \sum_{t=1}^{T} w_j \, t \, x_{jt} \tag{4.12a}$$

$$\text{s.t.} \qquad \sum_{t=1}^{T} x_{jt} = 1 \qquad \forall\, j \in N, \tag{4.12b}$$

$$\sum_{j \in N} \sum_{\tau=\max\{1,t-p_j+1\}}^{t} x_{j\tau} \leq 1 \qquad \forall\, t \in [T], \tag{4.12c}$$

$$\sum_{\tau=1}^{t+p_j} x_{j\tau} - \sum_{\tau=1}^{t} x_{i\tau} \leq 0 \qquad \forall\, (i,j) \in E_\wedge,\ \forall\, t \in [T - p_j], \tag{4.12d}$$

$$x_{jt} \geq 0 \qquad \forall\, j \in N,\ \forall\, t \in \{p_j, \ldots, T\}. \tag{4.12e}$$

There is a one-to-one correspondence between feasible integer solutions for LP (4.12) and feasible single machine schedules. Note that the size of LP (4.12) is not polynomial in the input size of the scheduling instance. One way to circumvent this problem is to consider an *interval-indexed* formulation instead, see, e.g., [73, 71] and Section 4.3.1. Typically, going from time-indexed LPs to interval-indexed LPs has to be paid for by an additional "$+\varepsilon$" in the approximation factor, see, e.g., [73, 71, 150]. Queyranne and Schulz [140] and Schulz [148] observed that any feasible solution $\overline{x}$ for LP (4.12) defines a feasible solution for the completion time LP (4.4) by setting $\overline{C}_j = \sum_{t=1}^{T} t \, \overline{x}_{jt}$. Thus, the integrality gap of the time-indexed LP (4.12) is at most 2.

Our approximation algorithms in Section 4.3 are based on the concept of α-*points*, which was introduced in [73, 134]. For a given $0 < \alpha \leq 1$ and a feasible solution $\overline{x}$ for LP (4.12), the α-*point* of a job j is the first integer point in time when an α-fraction of job j is completed in the solution $\overline{x}$.

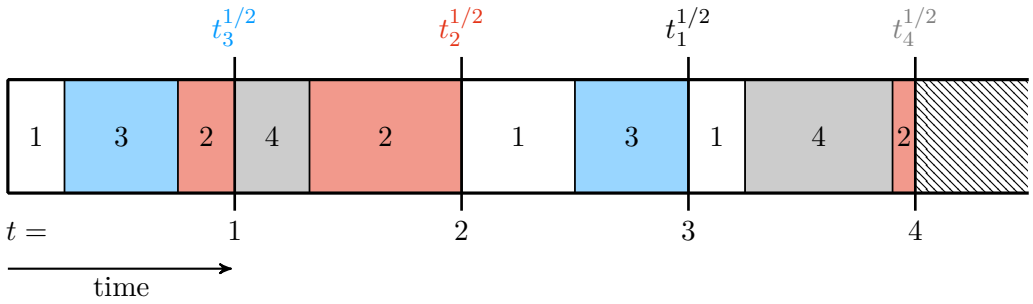

Fig. 4.3: A preemptive schedule for the instance and solution $\overline{x}$ in Example 4.3. Note that the α-points are independent of the ordering of the jobs within each interval $[t-1;t]$.

Definition 4.2 (α-Points)
Let $\overline{x}$ be a feasible solution for LP (4.12) and let $\alpha \in (0;1]$. The *α-point* of $j \in N$ is defined as

$$t_j^\alpha := \min\left\{ t \in [T] \,\middle|\, \sum_{\tau=1}^{t} \overline{x}_{j\tau} \geq \alpha \right\}. \tag{4.13}$$

We refer to [154] and references therein for several applications of α-point scheduling. For unit processing time jobs, the variables x_{jt} are defined for all $j \in N$ and $t \in [T]$. In this case, α-points can be illustrated by interpreting $\overline{x}$ as a preemptive single-machine schedule that processes an $\overline{x}_{jt}$-fraction of job j in the interval $[t-1;t]$. The ordering of the jobs in $\{j \in N \,|\, \overline{x}_{jt} > 0\}$ within each interval $[t-1;t]$ is arbitrary.

Example 4.3 (Preemptive Schedule and α-Points)
Consider an instance on four jobs $[4]$ with unit processing times and no precedence constraints. Let $T = p([4]) = 4$ and let $\overline{x}$ be defined via the following table:

$t =$	1	2	3	4
job 1	1/4	0	1/2	1/4
job 2	1/4	2/3	0	1/12
job 3	1/2	0	1/2	0
job 4	0	1/3	0	2/3

Note that the column sums and row sums are all equal to 1. So, $\overline{x}$ satisfies constraints (4.12b) and (4.12c), i.e., $\overline{x}$ is feasible for LP (4.12). Figure 4.3 illustrates a corresponding preemptive schedule. Recall that the ordering of the jobs within each interval $[t-1;t]$ is arbitrary. The $\frac{1}{2}$-points of the jobs are $t_1^{1/2} = 3$, $t_2^{1/2} = 2$, $t_3^{1/2} = 1$ and $t_4^{1/2} = 4$.

4.3 Time-Indexed Formulation

In this section, we use time-indexed formulations similar to LP (4.12) to derive approximation algorithms for bipartite AND/OR-scheduling (Definition 1.17) and all-but-one min-sum set cover (Definition 3.3). The algorithms are based on α-point scheduling (Definition 4.2).

4.3.1 Approximating Bipartite AND/OR-Scheduling

Consider an instance of $1 \,|\, \textit{ao-prec} = A \dot{\vee} B \,|\, \sum w_j C_j$ with precedence graph $G = (A \,\dot\cup\, B, E_\wedge \,\dot\cup\, E_\vee)$ where $E_\wedge \subseteq (A \times A) \cup (B \times B)$ and $E_\vee \subseteq A \times B$. W.l.o.g., suppose that $E_\wedge$ is transitively closed, i.e., $(i,j) \in E_\wedge$ and $(j,k) \in E_\wedge$ implies $(i,k) \in E_\wedge$. We may further assume that there are no redundant OR-precedence constraints, i.e., if $(a,b) \in E_\vee$ and $(a',a) \in E_\wedge$, then $(a',b) \notin E_\vee$. Otherwise, we could remove the arc (a,b) from $E_\vee$, since any feasible schedule has to schedule a' before a.

Let $\Delta := \max\{|\mathcal{P}(b)| \,|\, b \in B\}$ be the maximum number of OR-predecessors of a job in B. One can see that Δ is bounded from above by the cardinality of a maximum independent set in the induced subgraph on $E_\wedge \cap (A \times A)$. Note that Δ is often relatively small compared to the total number of jobs. For instance, if the precedence constraints are derived from an underlying graph, where the predecessors of each edge are its incident vertices (as in MSVC), then $\Delta \leq 2$. If $\mathcal{P}(b) = \emptyset$ for all $b \in B$, then the instance of bipartite AND/OR-scheduling is an instance of scheduling with AND-precedence constraints only. In this case, we set $\Delta = 1$ in the following.

Theorem 4.4
There is a 2Δ-approximation algorithm for bipartite AND/OR-scheduling with 0/1 processing times, i.e., for $1 \,|\, \textit{ao-prec} = A \dot{\vee} B,\ p_j \in \{0,1\} \,|\, \sum w_j C_j$. Moreover, there is a $(2\Delta + \varepsilon)$-approximation algorithm for $1 \,|\, \textit{ao-prec} = A \dot{\vee} B \,|\, \sum w_j C_j$ for any $\varepsilon > 0$.

First, we exhibit a randomized approximation algorithm for $1 \,|\, \textit{ao-prec} = A \dot{\vee} B,\ p_j \in \{0,1\} \,|\, \sum w_j C_j$, and then we show how to derandomize it. This proves the first part of Theorem 4.4. Recall from Figure 4.1 that bipartite AND/OR-scheduling with $p_a = 1$ for $a \in A$ and $p_b = 0$ for all $b \in B$ generalizes precedence-constrained MSSC.

A natural question that arises in the context of real-world scheduling problems is whether approximation guarantees for 0/1-problems still hold for arbitrary processing times. As observed by Munagala et al. [128], the natural extension of the Greedy algorithm for min-sum set cover still works, if the processing times of jobs in A are arbitrary, but all jobs in B have zero processing time, and there are no AND-precedence constraints. Once jobs in B have non-zero processing times, their analysis of the Greedy algorithm fails. Our algorithm can be extended to arbitrary processing times (which proves the second part of Theorem 4.4) and, additionally, release dates.

Note that the result of Theorem 4.4 improves upon the algorithm of [122] for precedence-constrained MSSC in two ways. First, the approximation factor of 2Δ does not depend on the total number of jobs, but on the maximum number of OR-predecessors of a job in B. In particular, we immediately obtain a 4-approximation algorithm for the special case of precedence-constrained MSVC. Secondly, our algorithm works for arbitrary processing times, additional AND-precedence constraints on $B \times B$, and it can be extended to non-trivial release dates $r_j \geq 0$ of the jobs. Note that, in general, the parameters Δ of Theorem 4.4 and $\sqrt{|U|}$ of [122] are incomparable. In most practically relevant instances, Δ should be considerably smaller than $\sqrt{|U|}$.

It is important to highlight that the approximation factor of 2Δ in Theorem 4.4 does not contradict the conjectured hardness of precedence-constrained MSSC stated in [122]. The set A in the reduction of [122] from the planted dense subgraph problem contains a job for every vertex and every edge of the random graph on m vertices. Each vertex-job consists of the singleton $\{0\}$, and each edge-job is a (random) subset of $[\ell] = \{1, \dots, \ell\}$ for some non-negative integer ℓ.[25] Each vertex-job is an AND-predecessor of the incident edge-jobs. Every element in $[\ell]$ appears in expectation in mp^2 many edge-jobs, where p is a carefully chosen probability. If we interpret this as a scheduling problem, we can delete the dummy element 0 from the instance. So the maximum indegree of a job in $B = [\ell]$ (maximum number of appearances of the element) is $\Delta \approx mp^2 \geq m^{1/4}$, see [122]. Hence the gap $\Omega(m^{1/8})$ in the reduction translates to a gap of $\Omega(\sqrt{\Delta})$ in our setting. Therefore, if the planted dense subgraph conjecture [24] holds true, then there is no $\mathcal{O}(\Delta^{1/3-\varepsilon})$-approximation algorithm for $1\,|\,ao\text{-}prec = A\dot{\vee}B\,|\,\sum w_jC_j$ for any $\varepsilon > 0$.

Note that, in the reduction from Hitting Set to $1\,|\,ao\text{-}prec = A\dot{\vee}B\,|\,\sum w_jC_j$, the parameter Δ equals the maximum cardinality of any set in the Hitting Set instance. Bar-Yehuda and Even [21] and Hochbaum [82] presented approximation algorithms for Hitting Set with a guarantee of Δ. Hence, the 2Δ-approximation of Theorem 4.4 does not contradict the hardness of obtaining a $(1-\varepsilon)\ln(|\mathcal{R}|)$-approximation for Hitting Set [39]. If the planted dense subgraph conjecture [24] is false, then constant-factor approximations for $1\,|\,ao\text{-}prec = A\dot{\vee}B\,|\,\sum w_jC_j$ with $E_\wedge \subseteq A \times A$ may be possible. However, the reduction from Hitting Set shows that, in general, we cannot get a constant-factor approximation if $E_\wedge \cap (B \times B) \neq \emptyset$, unless P = NP.

A New Algorithm for Precedence-Constrained MSSC. W.l.o.g., we may assume that $w_a = 0$ for all $a \in A$. Otherwise, we can shift a positive weight of a job in A to an additional successor in B with zero processing time. Further, we may assume that all data are integer and $p_j \geq 1$ for every job $j \in N$ that has no predecessors (otherwise such a job can be disregarded). So, no job can complete at time 0 in a feasible schedule. Let $p_j \in \{0, 1\}$ for all $j \in N$, and let $T = p(N)$ be the time horizon. We consider the time-indexed LP (4.12) with additional constraints corresponding to $E_\vee$.

[25] The definition of MSSC in [122] is reversed, i.e., the sets are ordered and an element is covered as soon as a set that contains the element appears in the linear ordering.

The resulting linear relaxation is

$$\min \quad \sum_{b\in B}\sum_{t=1}^{T} w_b\, t\, x_{bt} \tag{4.14a}$$

$$\text{s.t.} \quad \sum_{t=1}^{T} x_{jt} = 1 \qquad \forall\, j \in N, \tag{4.14b}$$

$$\sum_{j\in N}\sum_{\tau=t-p_j+1}^{t} x_{j\tau} \le 1 \qquad \forall\, t \in [T], \tag{4.14c}$$

$$\sum_{\tau=1}^{t+p_b} x_{b\tau} - \sum_{a\in\mathcal{P}(b)}\sum_{\tau=1}^{t} x_{a\tau} \le 0 \qquad \forall\, b \in B:\ \mathcal{P}(b) \neq \emptyset,\ \forall\, t \in [T-p_b], \tag{4.14d}$$

$$\sum_{\tau=1}^{t+p_j} x_{j\tau} - \sum_{\tau=1}^{t} x_{i\tau} \le 0 \qquad \forall\, (i,j) \in E_\wedge,\ \forall\, t \in [T-p_j], \tag{4.14e}$$

$$x_{jt} \ge 0 \qquad \forall\, j \in N,\ \forall\, t \in [T]. \tag{4.14f}$$

Constraints (4.14b) and (4.14c) ensure that each job is executed and no jobs overlap, respectively. Note that only jobs with $p_j = 1$ appear in (4.14c). Constraints (4.14d) and (4.14e) ensure OR- and AND-precedence constraints, respectively. Note that we can solve LP (4.14) in polynomial time, since $T \le n$.

Let $\overline{x}$ be an optimal fractional solution of LP (4.14). We call $\overline{C}_j = \sum_{t=1}^{T} t\,\overline{x}_{jt}$ the *fractional completion time* of $j \in N$. Note that $\sum_{j\in N} w_j\,\overline{C}_j$ is a lower bound on the objective value of an optimal integer solution, which corresponds to an optimal schedule. For $0 < \alpha \le 1$ and $j \in N$, we denote its α-point by t_j^α (Definition 4.2).

The algorithm works as follows and is summarized in Algorithm 5. First, we solve LP (4.14) to optimality, and let $\overline{x}$ be an optimal fractional solution (line 1). Then, we draw β at random from the interval $(0;1]$ with density function $f(\beta) = 2\beta$, and set $\alpha = \frac{\beta}{\Delta}$.[26] In line 5, we compute t_a^α and t_b^β for all jobs $a \in A$ and $b \in B$, respectively. We sort the jobs in order of non-decreasing values t_a^α ($a \in A$) and t_b^β ($b \in B$), and denote this total order by $\prec$. If there is $b \in B$ and $a \in \mathcal{P}(b)$ with $t_a^\alpha = t_b^\beta$, we set $a \prec b$. Similarly, we set $i \prec j$, if $(i,j) \in E_\wedge$ and $t_i^\alpha = t_j^\alpha$ (for $i,j \in A$) or $t_i^\beta = t_j^\beta$ (for $i,j \in B$).[27] All other ties are broken arbitrarily. The partial order $\prec'$ defined in the if-clauses in lines 11 and 14 ensures precisely these conditions. In line 19, the partial order $\prec'$ is extended to a total order $\prec$ on N, which can be done in polynomial time.

[26] Choosing α as a function of β is crucial in order to obtain a feasible schedule in the end. This together with (4.14d) ensures that at least one OR-predecessor of a job $b \in B$ completes early enough in the constructed schedule. The density function $f(\beta) = 2\beta$ is chosen to cancel out an unbounded term of $\frac{1}{\beta}$ in the expected value of the completion time of job b, as in [59, 149].

[27] Recall that $E_\wedge \subseteq (A \times A) \cup (B \times B)$, so $(i,j) \in E_\wedge$ implies $i,j \in A$ or $i,j \in B$.

Input: Instance $\mathcal{I}$ of $1 \,|\, ao\text{-}prec = A \dot{\vee} B,\ p_j \in \{0,1\} \,|\, \sum w_j C_j$
Output: Feasible schedule for $\mathcal{I}$

1 Solve LP (4.14) and let $\overline{x}$ be an optimum solution;
2 Draw β randomly from $(0;1]$ with density function $f(\beta) = 2\beta$ and set $\alpha = \frac{\beta}{\Delta}$;
3 **for** $j \in N$ **do**
4 Set $\mu \leftarrow \alpha$ (if $j \in A$) or $\mu \leftarrow \beta$ (if $j \in B$);
5 Compute $t_j^\mu = \min\{t \in [T] \,|\, \sum_{\tau=1}^{t} \overline{x}_{j\tau} \geq \mu\}$;
6 **end**
7 **for** $i \in N$ **do**
8 Set $\lambda \leftarrow \alpha$ (if $i \in A$) or $\lambda \leftarrow \beta$ (if $i \in B$);
9 **for** $j \in N$ **do**
10 Set $\mu \leftarrow \alpha$ (if $j \in A$) or $\mu \leftarrow \beta$ (if $j \in B$);
11 **if** $t_i^\lambda < t_j^\mu$ **then**
12 Set $i \prec' j$;
13 **end**
14 **if** $t_i^\lambda = t_j^\mu$ *and* $(i,j) \in E_\wedge \cup E_\vee$ **then**
15 Set $i \prec' j$;
16 **end**
17 **end**
18 **end**
19 Let $\prec$ be a total order of N such that $i \prec' j$ implies $i \prec j$;
20 **return** Schedule that processes jobs in order of $\prec$;

Algorithm 5: An algorithm for bipartite AND/OR-scheduling.

The next lemma shows that ordering jobs according to $\prec$ yields a feasible schedule whose expected objective value is at most 2Δ times the optimum.

Lemma 4.5
Algorithm 5 is a randomized 2Δ-approximation algorithm for $1 \,|\, ao\text{-}prec = A \dot{\vee} B,\ p_j \in \{0,1\} \,|\, \sum w_j C_j$.

Proof. Note that Algorithm 5 runs in polynomial time, since we can solve LP (4.14) in polynomial time. We first show that scheduling the jobs in order of $\prec$ yields a feasible schedule for any fixed $0 < \beta \leq 1$. Recall the definition of $\Delta = \max_{b \in B} |\mathcal{P}(b)|$. Let $0 < \beta \leq 1$ and set $\alpha = \alpha(\beta) = \frac{\beta}{\Delta}$ as in line 2 of Algorithm 5.

Note that $t_i^\alpha \leq t_j^\alpha$ for any $(i,j) \in E_\wedge \cap (A \times A)$ and $t_i^\beta \leq t_j^\beta$ for any $(i,j) \in E_\wedge \cap (B \times B)$ by (4.14e). The if-clauses in lines 11 and 14 in Algorithm 5 ensure $i \prec j$ whenever $(i,j) \in E_\wedge$. For $b \in B$ with $\mathcal{P}(b) \neq \emptyset$, constraint (4.14d) implies

$$\beta \leq \sum_{\tau=1}^{t_b^\beta} \overline{x}_{b\tau} \leq \sum_{a \in \mathcal{P}(b)} \sum_{\tau=1}^{t_b^\beta - p_b} \overline{x}_{a\tau}. \tag{4.15}$$

So, there is $a_b \in \mathcal{P}(b)$ such that

$$\sum_{\tau=1}^{t_b^\beta - p_b} \overline{x}_{a_b\tau} \geq \frac{\beta}{|\mathcal{P}(b)|} \geq \alpha. \tag{4.16}$$

Hence, $t_{a_b}^\alpha \leq t_b^\beta$, and thus $a_b \prec b$ by lines 11 and 14. So, $\prec$ satisfies all precedence constraints, and Algorithm 5 returns a feasible schedule.

As for the approximation factor, let $\nu_j(t) = \sum_{\tau=1}^{t} \overline{x}_{j\tau}$ be the fraction of job j that is completed by time $t \in \{0, \dots, T\}$, as in the proof of Theorem 3.13. Note that $\nu_j(0) = 0$, $\nu_j(T) = 1$ and $t_j^\gamma \leq t$ if $\gamma \leq \nu_j(t)$. For $0 < \gamma \leq 1$ and $j \in N$, we observe, similar to [61, 149], that

$$\begin{aligned} \int_0^1 t_j^\gamma \, d\gamma &= \sum_{t=1}^{T} \int_{\nu_j(t-1)}^{\nu_j(t)} t_j^\gamma \, d\gamma \leq \sum_{t=1}^{T} \Big(\nu_j(t) - \nu_j(t-1)\Big)\, t \\ &= \sum_{t=1}^{T} \left(\sum_{\tau=1}^{t} \overline{x}_{j\tau} - \sum_{\tau=1}^{t-1} \overline{x}_{j\tau} \right) t = \sum_{t=1}^{T} \overline{x}_{jt}\, t = \overline{C}_j. \end{aligned} \tag{4.17}$$

Note that there is no "idle time" on the machine in the optimal fractional solution $\overline{x}$, i.e., $\sum_{i \in N} \nu_i(t_j^\beta)\, p_i = t_j^\beta$ for all $i, j \in N$ and $0 < \beta \leq 1$. The completion times of the schedule returned by the algorithm for a specific realization of β are denoted by $C_j(\beta)$. Let $b \in B$ and $i \in N$ such that i precedes b in the schedule returned by Algorithm 5, i.e., $i \prec b$. By construction, we have $\alpha \leq \nu_i(t_b^\beta)$ (if $i \in A$) and $\alpha \leq \beta \leq \nu_i(t_b^\beta)$ (if $i \in B$), respectively. So,

$$\alpha\, C_b(\beta) = \sum_{i \preceq b} \alpha\, p_i \leq \sum_{i \preceq b} \nu_i(t_b^\beta)\, p_i \leq t_b^\beta. \tag{4.18}$$

Thus, the expected completion time of $b \in B$ is

$$\mathbb{E}[C_b(\beta)] = \int_0^1 f(\beta)\, C_b(\beta)\, d\beta \leq \int_0^1 f(\beta)\, \frac{\Delta}{\beta}\, t_b^\beta \, d\beta = 2\Delta \int_0^1 t_b^\beta \, d\beta \leq 2\Delta\, \overline{C}_b. \tag{4.19}$$

Since only jobs in B contribute to the objective function, this yields the claim. □

For fixed $\overline{x}$ and $0 < \beta \leq 1$, we call the schedule that orders the jobs according to $\prec$ the *β-schedule of $\overline{x}$*. Given $\overline{x}$ and $0 < \beta \leq 1$, we can construct the β-schedule in polynomial time. We derandomize Algorithm 5 by an observation similar to [26, 62]: List all possible schedules that occur as β goes from 0 to 1, and pick the best one. The next lemma shows that the number of different β-schedules is not too large.

Lemma 4.6
For every $\overline{x}$, there are $\mathcal{O}(n^2)$ different β-schedules.

Proof. Note that, as β goes from 0 to 1, $\alpha(\beta) = \frac{\beta}{\Delta}$ is a linear function with values from 0 to $\frac{1}{\Delta}$. We interpret the fractional solution $\overline{x}$ as a preemptive schedule where an $\overline{x}_{jt}$-fraction of job j is contiguously scheduled in time slot $[t-1;t]$. The order of jobs in the β-schedule of $\overline{x}$ only changes if the $\alpha(\beta)$-point or β-point of a job in A or B reaches a point when this job gets preempted, respectively. So, the number of different β-schedules is bounded from above by the number of preemptions in $\overline{x}$. Each job is preempted at most once within each time step. Recall that $T \in \mathcal{O}(n)$, since $p_j \in \{0,1\}$ for all $j \in N$. Hence, there are at most $n\,T \in \mathcal{O}(n^2)$ preemptions. □

Lemmas 4.5 and 4.6 together prove the first part of Theorem 4.4. Recall that $\Delta \leq 2$ for precedence constrained MSVC, which yields the following corollary of Theorem 4.4.

Corollary 4.7
There is a 4-approximation algorithm for precedence-constrained min-sum vertex cover.

Extensions of the Algorithm. If we use an interval-indexed LP instead of a time-indexed LP (see also [73, 71]), then Algorithm 5 can be generalized to arbitrary processing times. This proves the second part of Theorem 4.4. Let $\varepsilon' > 0$, and recall that all processing times are non-negative integers. Let $T = p(N)$ be the time horizon, and let L be minimal such that $(1+\varepsilon')^{L-1} \geq T$. Set $T_0 \coloneqq 1$, and let $T_\ell \coloneqq (1+\varepsilon')^{\ell-1}$ for every $\ell \in [L]$. We call $(T_{\ell-1}; T_\ell]$ the *ℓ-th interval* for $\ell \in [L]$. (The first interval is the singleton $(1;1] \coloneqq \{1\}$.) We introduce a binary variable $x_{j\ell}$ for every $j \in N$ and for every $\ell \in [L]$ that indicates whether or not job j completes in the ℓ-th interval. If we relax the integrality constraints on the variables, we obtain the following relaxation similar to LP (4.14) and [73, 71]:

$$\min \quad \sum_{b \in B} \sum_{\ell=1}^{L} w_b\, T_{\ell-1}\, x_{b\ell} \tag{4.20a}$$

$$\text{s.t.} \quad \sum_{\ell=1}^{L} x_{j\ell} = 1 \qquad \forall\, j \in N, \tag{4.20b}$$

$$\sum_{j \in N} \sum_{q=1}^{\ell} p_j\, x_{jq} \leq T_\ell \qquad \forall\, \ell \in [L], \tag{4.20c}$$

$$\sum_{q=1}^{\ell} x_{bq} - \sum_{a \in \mathcal{P}(b)} \sum_{q=1}^{\ell} x_{aq} \leq 0 \qquad \forall\, b \in B : \mathcal{P}(b) \neq \emptyset,\ \forall\, \ell \in [L], \tag{4.20d}$$

$$\sum_{q=1}^{\ell} x_{jq} - \sum_{q=1}^{\ell} x_{iq} \leq 0 \qquad \forall\, (i,j) \in E_\wedge,\ \forall\, \ell \in [L], \tag{4.20e}$$

$$x_{j\ell} \geq 0 \qquad \forall\, j \in N,\ \forall\, \ell \in [L] : T_{\ell-1} \geq p_j. \tag{4.20f}$$

Given ε', the size of LP (4.20) is polynomial, so we can solve it in polynomial time. Again, (4.20b) ensures that every job is executed. Constraints (4.20c) are valid for any feasible schedule, since the total processing time of all jobs that complete within the first ℓ intervals cannot exceed T_ℓ. Constraints (4.20d) and (4.20e) ensure that, at the end of each interval, the fractions of the jobs satisfy OR- and AND-precedence constraints, respectively.

Let $\overline{x}$ be an optimal fractional solution of LP (4.20), and let $\overline{C}_j = \sum_{\ell=1}^{L} T_{\ell-1}\, \overline{x}_{j\ell}$. Note that $\sum_{j\in N} w_j \overline{C}_j$ is a lower bound on the optimal objective value of an integer solution, which is a lower bound on the optimal value of a feasible schedule. Let $\ell_j^\alpha := \min\{\ell \in [L] \mid \sum_{q=1}^{\ell} \overline{x}_{jq} \geq \alpha\}$ be the *α-interval* of job $j \in N$. This generalizes the notion of α-points from Definition 4.2.

The algorithm for arbitrary processing times is similar to Algorithm 5. Informally, one only has to replace t_j^μ by ℓ_j^μ in Algorithm 5. Therefore, we refer to the new algorithm as *Algorithm 5 with intervals*. More precisely, the algorithm works as follows. Instead of solving LP (4.14) in line 1, we solve LP (4.20) with $\varepsilon' = \frac{\varepsilon}{2\Delta}$ to achieve an approximation factor of $2\Delta + \varepsilon$. Let $\overline{x}$ be an optimal solution of LP (4.20). We draw β at random from the interval $(0; 1]$ with density function $f(\beta) = 2\beta$, and set $\alpha = \frac{\beta}{\Delta}$. In line 5, we compute ℓ_a^α and ℓ_b^β for all jobs $a \in A$ and $b \in B$, respectively. We again sort the jobs in order of non-decreasing values ℓ_a^α ($a \in A$) and ℓ_b^β ($b \in B$) and denote this total order by $\prec$. If $\ell_a^\alpha = \ell_b^\beta$ for some $b \in B$ and $a \in \mathcal{P}(b)$, set $a \prec b$. Similarly, set $i \prec j$ if $(i, j) \in E_\wedge$ and $\ell_i^\alpha = \ell_j^\alpha$ (for $i, j \in A$) or $\ell_i^\beta = \ell_j^\beta$ (for $i, j \in B$). All other ties are broken arbitrarily. Finally, the jobs are scheduled in the order of $\prec$.

Lemma 4.8
For any $\varepsilon > 0$, Algorithm 5 with intervals is a randomized $(2\Delta + \varepsilon)$-approximation for $1 \,|\, ao\text{-}prec = A \dot{\vee} B \,|\, \sum w_j C_j$.

Proof. The proof is similar to the proof of Lemma 4.5. Note that $\varepsilon' = \frac{\varepsilon}{2\Delta}$ is polynomial in the input. So, we can solve LP (4.20) in polynomial time and, thus, Algorithm 5 with intervals runs in polynomial time. Let $\overline{x}$ be an optimal solution of LP (4.20) and let $\overline{C}_j = \sum_{\ell=1}^{L} T_{\ell-1}\, \overline{x}_{j\ell}$ be the fractional completion time of $j \in N$.

Let $0 < \beta \leq 1$ and $\alpha = \alpha(\beta) = \frac{\beta}{\Delta}$. For any $(i, j) \in E_\wedge$, observe that $\ell_i^\alpha \leq \ell_j^\alpha$ (for $i, j \in A$) and $\ell_i^\beta \leq \ell_j^\beta$ (for $i, j \in B$) due to (4.20e). Constraints (4.20d) imply that, for every $b \in B$ with $\mathcal{P}(b) \neq \emptyset$, there is $a_b \in \mathcal{P}(b)$ that satisfies

$$\sum_{q=1}^{\ell_b^\beta} \overline{x}_{a_b q} \geq \frac{\beta}{|\mathcal{P}(b)|} \geq \alpha. \tag{4.21}$$

So, $\ell_{a_b}^\alpha \leq \ell_b^\beta$. Hence, ordering the jobs according to $\prec$ respects all precedence constraints due to the if-clauses in lines 11 and 14 in Algorithm 5 (with intervals). Thus, the schedule returned by Algorithm 5 with intervals is feasible.

As for the approximation factor, let $\nu_j(\ell) = \nu_j(T_\ell) = \sum_{q=1}^{\ell} \overline{x}_{jq}$ be the fraction of job j that is completed until time T_ℓ for $\ell \in \{0, \dots, L\}$. Note that $\nu_j(0) = 0$, $\nu_j(L) = 1$ and $T_{\ell_j^\gamma - 1} \leq T_{\ell-1}$ for $\gamma \leq \nu_j(\ell)$. We obtain

$$\begin{aligned} \int_0^1 T_{\ell_j^\gamma - 1}\, d\gamma &= \sum_{\ell=1}^{L} \int_{\nu_j(\ell-1)}^{\nu_j(\ell)} T_{\ell_j^\gamma - 1}\, d\gamma \leq \sum_{\ell=1}^{L} \Big(\nu_j(\ell) - \nu_j(\ell-1)\Big)\, T_{\ell-1} \\ &= \sum_{\ell=1}^{L} \left(\sum_{q=1}^{\ell} \overline{x}_{jq} - \sum_{q=1}^{\ell-1} \overline{x}_{jq} \right) T_{\ell-1} = \sum_{\ell=1}^{L} \overline{x}_{j\ell}\, T_{\ell-1} = \overline{C}_j. \end{aligned} \tag{4.22}$$

Let $b \in B$ and $i \in N$ with $i \prec b$. Then, $\alpha \leq \nu_i(\ell_b^\beta)$ (if $i \in A$) and $\alpha \leq \beta \leq \nu_i(\ell_b^\beta)$ (if $i \in B$), respectively. Further, (4.20c) implies

$$\alpha \sum_{i \preceq b} p_i \leq \sum_{i \preceq b} \nu_i(\ell_b^\beta)\, p_i = \sum_{i \preceq b} \sum_{q=1}^{\ell_b^\beta} p_i\, \overline{x}_{iq} \leq \sum_{i \in N} \sum_{q=1}^{\ell_b^\beta} p_i\, \overline{x}_{iq} \leq T_{\ell_b^\beta} \tag{4.23}$$

Let $C_j(\beta)$ be the completion time of job j in the schedule returned by the algorithm for a realization of β. It holds $C_b(\beta) \leq \sum_{i \preceq b} p_i$ for all $b \in B$. So, $C_b(\beta) \leq \sum_{i \preceq b} p_i \leq \frac{1}{\alpha} T_{\ell_b^\beta}$ by (4.23). If we draw β randomly from $(0; 1]$ with density function $f(\beta) = 2\beta$, then the expected completion time of $b \in B$ is

$$\begin{aligned} \mathbb{E}[C_b(\beta)] &\leq \int_0^1 f(\beta)\, \frac{\Delta}{\beta}\, T_{\ell_b^\beta}\, d\beta = \int_0^1 f(\beta)\, \frac{\Delta}{\beta}\, (1+\varepsilon')\, T_{\ell_b^\beta - 1}\, d\beta \\ &= 2\Delta(1+\varepsilon') \int_0^1 T_{\ell_b^\beta - 1}\, d\beta \leq (2\Delta + \varepsilon)\, \overline{C}_b, \end{aligned} \tag{4.24}$$

where the last inequality is due to (4.22) and the choice of ε'. Since only jobs in B contribute to the objective value, this proves the claim. □

The algorithm can be derandomized similar to Lemma 4.6. We interpret $\overline{x}$ as a preemptive schedule that assigns jobs to intervals. Note that each job is preempted at most once per time interval $(T_{\ell-1}; T_\ell]$. So, the number of β-schedules is bounded from above by $n\,L$, which is polynomially bounded in the input size. This proves the second part of Theorem 4.4.

Algorithm 5 can be further extended to cope also with non-trivial release dates. To do so, we add constraints to LP (4.14) and LP (4.20) that ensure that no job completes too early. More precisely, fix $x_{jt} = 0$ for all $j \in N$ and $t < r_j + p_j$ in LP (4.14), and $x_{j\ell} = 0$ for all $j \in N$ and $T_{\ell-1} < r_j + p_j$ in LP (4.20), respectively. When scheduling the jobs according to $\prec$, we might have to add idle time in order to respect the release dates. This increases the approximation factor slightly.

Lemma 4.9
There is a $(2\Delta+2)$*- and* $(2\Delta+2+\varepsilon)$*-approximation algorithm for* $1 \mid r_j$*,* $\textit{ao-prec} = A \dot{\vee} B$*,* $p_j \in \{0,1\} \mid \sum w_j C_j$ *and* $1 \mid r_j$*,* $\textit{ao-prec} = A \dot{\vee} B \mid \sum w_j C_j$*, respectively.*

Proof. The proof works similar to the proofs of Lemmas 4.5 and 4.8 and is inspired by [73]. We formulate the proof only for the case of arbitrary processing times. It can be easily adapted to the 0/1 case, by informally replacing T_ℓ and ℓ by t, and by setting $\varepsilon' = 0$. Similar to before, ordering the jobs according to $\prec$ respects all precedence constraints. If we add idle time where necessary, such that each job starts only after its release date, the schedule returned by the algorithm is feasible.

As for the approximation factor, assume that there are jobs with non-trivial release dates, and let $0 < \beta \leq 1$. Let $b \in B$ and $i \in A$ with $i \prec b$. Then, $\beta > 0$ implies $\alpha = \frac{\beta}{\Delta} > 0$, and, thus, $T_{\ell_b^\beta} \geq T_{\ell_i^\alpha} \geq r_i$. Similarly, $T_{\ell_b^\beta} \geq T_{\ell_i^\beta} \geq r_i$ for $i \in B$ and $i \prec b$. Hence, $T_{\ell_b^\beta} \geq \max_{i \preceq b} r_i$. The completion time of $b \in B$ in the schedule returned by the algorithm for a specific realization of β is

$$C_b(\beta) \leq \max_{i \preceq b} r_i + \sum_{i \preceq b} p_i \leq T_{\ell_b^\beta} + \frac{1}{\alpha} T_{\ell_b^\beta} = \left(1 + \frac{\Delta}{\beta}\right) T_{\ell_b^\beta} \leq \frac{1+\Delta}{\beta} T_{\ell_b^\beta}, \tag{4.25}$$

where the second inequality follows from (4.23). Note that (4.23) is not affected by release dates, since we only bound $\alpha \leq \nu_i(\ell_b^\beta) = \sum_{q=1}^{\ell_b^\beta} \overline{x}_{iq}$ and use constraint (4.20c). If we choose $\varepsilon' = \frac{\varepsilon}{2\Delta+2}$, then similar to (4.24), the expected value of the completion time of $b \in B$ is

$$\begin{aligned} \mathbb{E}[C_b(\beta)] &\leq \int_0^1 f(\beta) \frac{1+\Delta}{\beta} T_{\ell_b^\beta} \, d\beta = 2(\Delta+1)(1+\varepsilon') \int_0^1 T_{\ell_b^\beta - 1} \, d\beta \\ &\leq (2\Delta + 2 + \varepsilon) \, \overline{C}_b. \end{aligned} \tag{4.26}$$

Note that only jobs in B contribute to the objective value. We can derandomize similar to Lemma 4.6, since each job only gets preempted at most once per time slot/interval, also in the presence of release dates. This proves the claim. □

4.3.2 A 4-Approximation for All-But-One Min-Sum Set Cover

Recall that we can model GMSSC (Definition 1.10) as single-machine scheduling problem to minimize the sum of weighted completion times with job set $N = A \dot{\cup} B$, processing times $p_j \in \{0,1\}$, and certain precedence requirements $\kappa(b)$ on the jobs in B. We denote the set of predecessors of a job $b \in B$ by $\mathcal{P}(b)$. In this section, we use Algorithm 5 to obtain a 4-approximation algorithm for all-but-one min-sum set cover and the corresponding more general scheduling problem. Recall that all-but-one MSSC is the special case of generalized min-sum set cover where $\kappa(b) = \max\{1, |\mathcal{P}(b)| - 1\}$ for all $b \in B$. So, each job in B requires all but one of its predecessors to be completed before it can start, unless it has only one predecessor (Definition 3.3).

The linear programming formulation is similar to LP (4.14) and is based on the following observation. Suppose we want to schedule a job $b \in B$ with $|\mathcal{P}(b)| \geq 2$ at time $t \geq 0$. Then, at least $|\mathcal{P}(b)| - 1$ of its predecessors have to be completed before time t. Equivalently, for each pair of distinct $i, j \in \mathcal{P}(b)$, at most one of the two jobs i, j may complete after t. This gives the following linear relaxation with the same time-indexed variables as before and time horizon $T = p(N) \leq n$:

$$\min \quad \sum_{b \in B} \sum_{t=1}^{T} w_b \, t \, x_{bt} \tag{4.27a}$$

$$\text{s.t.} \quad \sum_{t=1}^{T} x_{jt} = 1 \quad \forall \, j \in N, \tag{4.27b}$$

$$\sum_{j \in N} \sum_{\tau=t-p_j+1}^{t} x_{j\tau} \leq 1 \quad \forall \, t \in [T], \tag{4.27c}$$

$$\sum_{\tau=1}^{t+p_b} x_{b\tau} - \sum_{\tau=1}^{t} (x_{i\tau} + x_{j\tau}) \leq 0 \quad \forall \, b \in B, \ \forall i, j \in \mathcal{P}(b), \ \forall \, t \in [T - p_b], \tag{4.27d}$$

$$\sum_{\tau=1}^{t+p_b} x_{b\tau} - \sum_{\tau=1}^{t} x_{i\tau} \leq 0 \quad \forall \, b \in B : \mathcal{P}(b) = \{i\}, \ \forall \, t \in [T - p_b], \tag{4.27e}$$

$$x_{jt} \geq 0 \quad \forall \, j \in N, \ \forall \, t \in [T]. \tag{4.27f}$$

Constraints (4.27b) and (4.27c) again ensure that each job is processed and no jobs overlap, respectively. Note that only jobs with non-zero processing time contribute to (4.27c). If $|\mathcal{P}(b)| = 1$, then (4.27e) dominates (4.27d). It ensures that the unique predecessor of $b \in B$ is completed before b starts. Note that this is a classical AND-precedence constraint, which does not affect the approximation factor.

For $|\mathcal{P}(b)| \geq 2$, (4.27d) models the above observation. Suppose at most $|\mathcal{P}(b)| - 2$ predecessors of b complete before t. Then, there are distinct $i, j \in \mathcal{P}(b)$ such that $\sum_{\tau=1}^{t}(x_{i\tau} + x_{j\tau}) = 0 \geq \sum_{\tau=1}^{t+p_b} x_{b\tau}$, so b cannot complete by time $t + p_b$.

LP (4.27) can be solved in polynomial time. Similar to Algorithm 5 and its extensions in Section 4.3.1, we first solve LP (4.27) and let $\overline{x}$ be an optimal fractional solution. We then draw β randomly from the interval $(0; 1]$ with density function $f(\beta) = 2\beta$, and schedule the jobs in A and B in order of non-decreasing $\frac{\beta}{2}$-points and β-points, respectively.[28] Again, we break ties consistently with the precedence constraints. For convenience, we do not make this algorithm explicit, as it suffices to use Algorithm 5 with LP (4.27) and $\alpha = \frac{\beta}{2}$.

[28] Choosing $\frac{\beta}{2}$ for the jobs in A ensures that at most one of the predecessors of a job $b \in B$ is scheduled after b in the constructed schedule

Theorem 4.10
There is a 4-approximation algorithm for all-but-one min-sum set cover.

Proof. The proof is fairly similar to the proof of Lemma 4.5. Let $\overline{x}$ be an optimal fractional solution to LP (4.27) and let $\overline{C}_j = \sum_{t=1}^{T} t\,\overline{x}_{jt}$ be the fractional completion time of job j. Let $b \in B$ with $|\mathcal{P}(b)| \geq 2$, and let $0 < \beta \leq 1$ and $\alpha = \frac{\beta}{2}$. Then, at least $|\mathcal{P}(b)| - 1$ predecessors of b have their α-point before time t_b^β. Suppose not, and let $i, j \in \mathcal{P}(b)$ such that $t_i^\alpha, t_j^\alpha > t_b^\beta$. It holds

$$\sum_{\tau=1}^{t_b^\beta - p_b} (\overline{x}_{i\tau} + \overline{x}_{j\tau}) \leq \sum_{\tau=1}^{t_b^\beta} \overline{x}_{i\tau} + \sum_{\tau=1}^{t_b^\beta} \overline{x}_{j\tau} < \alpha + \alpha = \beta \leq \sum_{\tau=1}^{t_b^\beta} \overline{x}_{b\tau}, \tag{4.28}$$

which contradicts (4.27d). So, the schedule returned by Algorithm 5 with LP (4.27) and $\alpha = \frac{\beta}{2}$ is feasible.

Similar to (4.17) in the proof of Lemma 4.5, we observe that $\int_0^1 t_b^\beta \, d\beta \leq \overline{C}_b$. Let $C_j(\beta)$ be the completion time of j in the resulting schedule for a realization of β, and let $\prec$ be the order of the jobs in this schedule. Observe that $C_b(\beta) = \sum_{i \preceq b} p_i \leq \frac{t_b^\beta}{\alpha} = \frac{2}{\beta} t_b^\beta$, as in (4.18). If we draw β randomly from $(0; 1]$ with density function $f(\beta) = 2\beta$, the expected completion time of $b \in B$ is

$$\mathbb{E}[C_b(\beta)] = \int_0^1 f(\beta) \frac{2}{\beta} t_b^\beta \, d\beta = 4 \int_0^1 t_b^\beta \, d\beta \leq 4\,\overline{C}_b. \tag{4.29}$$

Since only jobs in B contribute to the objective function, this gives a randomized 4-approximation algorithm for all-but-one min-sum set cover. The algorithm can be derandomized similar to Lemma 4.6, which proves the claim. □

Note that the algorithm also works if jobs in B have unit processing time. We can extend it to arbitrary processing times and release dates similar to Lemmas 4.8 and 4.9. If we choose $\varepsilon' = \frac{\varepsilon}{4}$ and solve the corresponding interval-indexed formulation instead of LP (4.27), we obtain a $(4 + \varepsilon)$-approximation for any $\varepsilon > 0$ for the corresponding scheduling problem. Again, AND-precedence constraints do not affect the approximation factor, similar to Lemmas 4.5 and 4.8. The following lemma shows that the analysis in Theorem 4.10 is tight.

Lemma 4.11
The integrality gap of LP (4.27) is equal to 4.

Proof. Note that Theorem 4.10 yields an upper bound of 4 on the integrality gap. As for the lower bound, let $n \geq 4$ be an even integer. Let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$ with $\mathcal{P}(b_\ell) = A \setminus \{a_\ell\}$ and $\kappa(b_\ell) = n - 2$ for all $\ell \in [n]$. Figure 4.4 depicts the precedence graph of this instance. Further, let $p_a = w_b = 1$ and $w_a = p_b = 0$ for all $a \in A$ and $b \in B$. Note that this is an instance of all-but-one min-sum set cover.

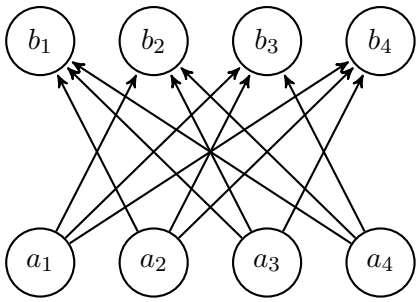

Fig. 4.4: Precedence graph of the instance (here with $n = 4$) for which the integrality gap of LP (4.27) approaches 4.

An optimal solution is to schedule the jobs in A in any arbitrary order, and then process each job in B as early as possible. W.l.o.g., we assume that the jobs in A are scheduled such that job $a_\ell \in A$ completes at time ℓ. So, jobs $b_{n-1}, b_n \in B$ both can complete at time $n-2$, and all other jobs in B complete at time $n-1$. The objective value of this schedule is equal to $2(n-2) + (n-2)(n-1) = (n-2)(n+1)$.

On the other hand, consider the following fractional solution with $\overline{x}_{at} = \frac{1}{n}$ for all $a \in A$ and $1 \leq t \leq n = T$. For $b \in B$, set $\overline{x}_{bt} = \frac{2}{n}$ for $1 \leq t \leq \frac{n}{2}$ and $\overline{x}_{bt} = 0$ else. One can easily verify that this solution is feasible for LP (4.27). (Note that constraints (4.27e) do not appear, since $|\mathcal{P}(b)| \geq 2$ for all $b \in B$.) Its objective value is equal to

$$\sum_{b \in B} \sum_{t=1}^{n} t\, \overline{x}_{bt} = n \sum_{t=1}^{\frac{n}{2}} t\, \frac{2}{n} = \frac{n}{2}\left(\frac{n}{2} + 1\right) = \frac{1}{4} n(n+2). \tag{4.30}$$

So, the integrality gap of LP (4.27) approaches 4 as n goes to infinity. □

Lemma 4.11 shows that the approximation ratio of 4 in Theorem 4.10 is the best we can get if we stick to LP (4.27). However, this does not rule out the existence of, e.g., combinatorial algorithms or stronger formulations for all-but-one MSSC, which could give approximation guarantees better than 4.

4.4 Linear Ordering Formulation

In this section, we show that the natural LP formulation in linear ordering variables exhibits an integrality gap (Definition 1.6) that is linear in the number of jobs when OR-precedence constraints are involved, even on instances where $\Delta = \max_{b \in B} |\mathcal{P}(b)| = 2$. We first restrict to bipartite precedence graphs of the form $G = (A \dot\cup B, E_\vee)$ with $E_\vee \subseteq A \times B$ in Section 4.4.1. We present a class of facet-defining inequalities and show that the integrality gap of the LP formulation remains unbounded, even if we add these facet-defining constraints. In Section 4.4.2, we extend the results to general acyclic precedence graphs using the notion of bottlenecks (Definition 4.15).

4.4.1 Bipartite OR-Precedence Constraints

Let $G = (A \dot\cup B, E_\vee)$ with $E_\vee \subseteq A \times B$ be a bipartite precedence graph. W.l.o.g., we may assume that every job in B has a predecessor, i.e., $\mathcal{P}(b) \neq \emptyset$ for all $b \in B$. Recall the linear ordering formulation LP (4.8) of Potts [137]. A nice feature of this formulation is that we can model OR-precedence constraints in a very intuitive way with constraints $\sum_{i\in\mathcal{P}(j)} \delta_{ij} \geq 1$ for all $j \in N$ with $\mathcal{P}(j) \neq \emptyset$. This gives the following polynomial size linear ordering relaxation for OR-scheduling:

$$\min \quad \sum_{j\in N}\sum_{i\in N} w_j\, p_i\, \delta_{ij} \tag{4.31a}$$

$$\text{s.t.} \quad \delta_{ij} + \delta_{ji} = 1 \qquad \forall\, i,j \in N :\ i \neq j \tag{4.31b}$$

$$\delta_{ij} + \delta_{jk} + \delta_{ki} \geq 1 \qquad \forall\, i,j,k \in N \tag{4.31c}$$

$$\sum_{i\in\mathcal{P}(j)} \delta_{ij} \geq 1 \qquad \forall\, j \in N :\ \mathcal{P}(j) \neq \emptyset, \tag{4.31d}$$

$$\delta_{ii} = 1 \qquad \forall\, i \in N, \tag{4.31e}$$

$$\delta_{ij} \geq 0 \qquad \forall\, i,j \in N. \tag{4.31f}$$

Note that (4.31d) only applies to jobs in B if the precedence graph is bipartite. We set $\delta_{ii} = 1$ in (4.31e), so the completion time of job j is $C_j = \sum_{i\in N} p_i\, \delta_{ij}$. Every feasible single-machine schedule without idle time corresponds to a feasible integer solution of LP (4.31), and vice versa. Recall that, for AND-scheduling, the linear ordering formulation LP (4.8) has an integrality gap of 2, see Section 4.2.2. However, in the presence of OR-precedence constraints, the gap of LP (4.31) grows linearly in the number of jobs, even if $\Delta = 2$, i.e., if every job in B has at most two predecessors.

Lemma 4.12
There is a family of instances such that the integrality gap of LP (4.31) is $\Omega(n)$.

Proof. Let $n \in \mathbb{N}$ be a multiple of 3. Consider an instance that consists of $\ell = \frac{n}{3}$ copies of the following directed graph on three jobs $\{i,j,k\}$. The processing times and weights are equal to $p_i = p_k = 1$, $p_j = 0$ and $w_i = w_k = 0$, $w_j = 1$. The jobs i,k do not have predecessors, and $\mathcal{P}(j) = \{i,k\}$. We indicate the job sets of copy $q \in [\ell]$ by $N_q = \{i_q, j_q, k_q\}$, and set $N = N_1 \cup \cdots \cup N_\ell$. That is, $A = \{i_1,\ldots,i_\ell,k_1,\ldots,k_\ell\}$ and $B = \{j_1,\ldots,j_\ell\}$. Note that $|N| = 3\ell = n$ and $\Delta = 2$, see Figure 4.5 for an example.

Any feasible schedule has to schedule i_q or k_q before j_q for all $q \in [\ell]$. Further, any optimal schedule will always schedule j_q immediately after i_q or k_q, whichever completes first. Since $p_{i_q} = p_{k_q} = 1$ for all $q \in [\ell]$, it does not matter whether i_q or k_q precedes j_q, and the order of the copies does not matter either. So, an optimal integer solution has an objective value of

$$\sum_{q=1}^{\ell} q = \frac{\ell(\ell+1)}{2} \in \Omega(n^2). \tag{4.32}$$

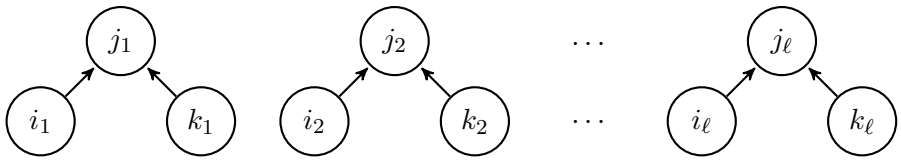

Fig. 4.5: An OR-scheduling instance for which LP (4.31) exhibits an integrality gap that is linear in the number of jobs. The processing times and weights are $p_{j_q} = 0$, $p_{i_q} = p_{k_q} = 1$ and $w_{j_q} = 1$, $w_{i_q} = w_{k_q} = 0$ for all $q \in [\ell]$.

On the other hand, consider the following fractional solution. For all $q \in [\ell]$ set $\overline{\delta}_{i_q j_q} = \overline{\delta}_{k_q j_q} = \overline{\delta}_{i_q k_q} = \frac{1}{2}$. For distinct $q_1, q_2 \in [\ell]$, set $\overline{\delta}_{i_{q_1} i_{q_2}} = \overline{\delta}_{k_{q_1} k_{q_2}} = \overline{\delta}_{j_{q_1} j_{q_2}} = \overline{\delta}_{i_{q_1} k_{q_2}} = \frac{1}{2}$ and $\overline{\delta}_{i_{q_1} j_{q_2}} = \overline{\delta}_{k_{q_1} j_{q_2}} = 0$. Further, let $\overline{\delta}_{ij} = 1 - \overline{\delta}_{ji}$ and $\overline{\delta}_{ii} = 1$ for all distinct $i, j \in N$. Note that $\overline{\delta}$ is a feasible solution of LP (4.31).

As for the objective value, recall that $p_{j_q} = w_{i_q} = w_{k_q} = 0$ and $p_{i_q} = p_{k_q} = w_{j_q} = 1$ for all $q \in [\ell]$. So, the only variables that contribute to the objective function with a non-zero coefficient are $\delta_{i_{q_1} j_{q_2}}$ and $\delta_{k_{q_1} j_{q_2}}$ for all $q_1, q_2 \in [\ell]$. The coefficient of these variables is equal to 1. Further, $\overline{\delta}_{i_{q_1} j_{q_2}} = \overline{\delta}_{k_{q_1} j_{q_2}} = 0$ for distinct $q_1, q_2 \in [\ell]$. Hence, the objective value of $\overline{\delta}$ is equal to

$$\sum_{q_1=1}^{\ell} \sum_{q_2=1}^{\ell} \left(\overline{\delta}_{i_{q_1} j_{q_2}} + \overline{\delta}_{k_{q_1} j_{q_2}} \right) = \sum_{q=1}^{\ell} \left(\overline{\delta}_{i_q j_q} + \overline{\delta}_{k_q j_q} \right) = \sum_{q=1}^{\ell} \left(\frac{1}{2} + \frac{1}{2} \right) = \ell \in \mathcal{O}(n). \quad (4.33)$$

Since $\overline{\delta}$ is feasible, the optimal objective value of LP (4.31) is $\mathcal{O}(n)$. The gap of (4.32) and (4.33) and, thus, the integrality gap of LP (4.31) is $\Omega(n)$. □

Note that the instance in the proof of Lemma 4.12 satisfies $|\mathcal{P}(b)| \leq 2$ for all $b \in B$. For this special case, we exhibit facet-defining inequalities in the remainder of this section. If $|\mathcal{P}(b)| \leq 2$ for all $b \in B$, then LP (4.31) can be written as

$$\begin{aligned}
\min \quad & \sum_{j \in N} \sum_{i \in N} w_j \, p_i \, \delta_{ij} && && (4.34a)\\
\text{s.t.} \quad & \delta_{ij} + \delta_{ji} = 1 && \forall \, i, j \in N : \, i \neq j, && (4.34b)\\
& \delta_{ij} + \delta_{jk} + \delta_{ki} \geq 1 && \forall \, i, j, k \in N, && (4.34c)\\
& \delta_{ab} + \delta_{a'b} \geq 1 && \forall \, b \in B : \, \mathcal{P}(b) = \{a, a'\}, && (4.34d)\\
& \delta_{ij} = 1 && \forall \, i, j \in N : \; i = j \text{ or } \mathcal{P}(j) = \{i\}, && (4.34e)\\
& \delta_{ij} \geq 0 && \forall \, i, j \in N. && (4.34f)
\end{aligned}$$

Note that constraints (4.34b) and (4.34c) coincide with the corresponding constraints in LP (4.31). Constraints (4.34d) model the OR-precedence constraints for jobs $b \in B$ with $|\mathcal{P}(b)| = 2$. For $b \in B$ with $|\mathcal{P}(b)| = 1$, the corresponding OR-precedence constraint is included in (4.34e). The main result in this section is a characterization of facet-defining constraints for the linear ordering LP (4.34).

Theorem 4.13
For all $b \in B$ and $\mathcal{P}(b) = \{a, a'\}$, the constraints

$$\delta_{aa'} + \delta_{a'b} \geq 1 \tag{4.35}$$

are valid for the integer hull of LP (4.34). Moreover, if they are tight, then they are either facet-defining or equality holds for all feasible integer solutions of LP (4.34).

It can easily be verified that the fractional solution in the proof of Lemma 4.12 satisfies (4.35), and is feasible for LP (4.34). Hence the integrality gaps of LPs (4.31) and (4.34) remain linear in the number of jobs, even if we add constraints (4.35). The remainder of this section is dedicated to prove Theorem 4.13.

In the following, we interchangeably use δ to denote a total order of the jobs, i.e., a single-machine schedule, and the corresponding 0/1 vector of the linear ordering polytope LP (4.34). First, we discuss why constraints (4.35) are valid for any feasible schedule. Note that any schedule (whether it is feasible w.r.t. the precedence constraints or not) violates at most one of the constraints $\delta_{aa'} + \delta_{a'b} \geq 1$ or $\delta_{a'a} + \delta_{ab} \geq 1$, since $\delta_{aa'} + \delta_{a'a} = 1$ by (4.34b). Hence, in order for one of these inequalities to be violated, we need $\delta_{ab} = \delta_{a'b} = 0$. But then b precedes a and a', so the precedence constraints of b are violated, and the schedule is infeasible. Note that constraints (4.35) together with (4.34b) dominate constraints (4.34d).

To prove the second part of Theorem 4.13, we make use of the following polyhedral observation. Let Q be the integer hull of the feasible region of LP (4.34) if we replace constraints (4.34e) by (4.31e), i.e., we remove all OR-precedence constraints with only one OR-predecessor from the precedence graph G. That is,

$$Q := \operatorname{conv}(\{\delta \in \{0,1\}^{n^2} \mid (4.34b),(4.34c),(4.34d),(4.31e)\}). \tag{4.36}$$

The precedence graph of the resulting OR-scheduling instance is denoted by G'. Note that this instance satisfies $|\mathcal{P}(b)| \in \{0, 2\}$ for all $b \in B$.

Clearly, all feasible vectors $\delta \in Q$ satisfy $0 \leq \delta_{ij} \leq 1$ for all $i, j \in N$. That is, for all distinct $i, j \in N$, the removed constraint of (4.34e) defines a supporting hyperplane at Q, which we call H_{ij}. In particular, for any facet F of Q, either $F \cap H_{ij} \in \{\emptyset, Q \cap H_{ij}\}$ or $F \cap H_{ij}$ is a facet of $Q \cap H_{ij}$. So, in order to prove Theorem 4.13, it suffices to show that constraints (4.35) are facet-defining for Q. We do so by exhibiting $\dim(Q)$ affinely independent feasible vectors of Q that satisfy (4.35) with equality. Similar to [69, 140], it is easy to see that Q is not contained in any lower dimensional affine subspace than the one spanned by constraints (4.34b) and (4.31e). So, the dimension of Q is equal to $d = \frac{n(n-1)}{2}$. The following lemma completes the proof of Theorem 4.13.

Lemma 4.14
Constraints (4.35) are facet-defining for Q.

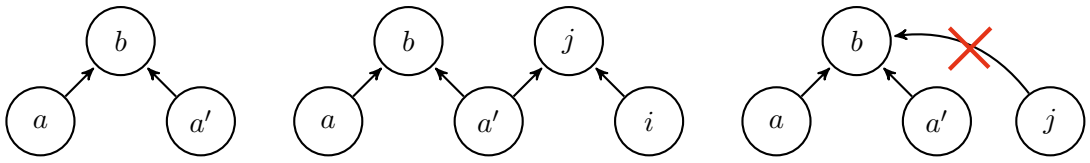

Fig. 4.6: Precedence graphs for $n = 3$ (left), and for $n \geq 4$ with $B \setminus \{b\} \neq \emptyset$ (middle) and $B \setminus \{b\} = \emptyset$ (right). The crossed out arc cannot occur, since $|\mathcal{P}(b)| = 2$ by assumption.

Proof. Let $b \in B$ with $\mathcal{P}(b) = \{a, a'\}$. We prove the statement by exhibiting d affinely independent integer feasible points for Q that satisfy $\delta_{aa'} + \delta_{a'b} = 1$. Recall that the instance on G' that corresponds to Q satisfies $|\mathcal{P}(b')| \in \{0, 2\}$ for all $b' \in B$. The proof goes by induction on the number of jobs n.

The base case is $n = 3$, i.e., $d = 3$. There is only one possible precedence graph G' that can occur, see Figure 4.6 (left). All feasible schedules for G' are $a \to a' \to b$, $a' \to a \to b$, $a \to b \to a'$ and $a' \to b \to a$. Obviously, all but the first schedule, $a \to a' \to b$, satisfy $\delta_{aa'} + \delta_{a'b} = 1$, and their respective δ-vectors are affinely independent. So, the claim holds for $n = 3$.

By induction hypothesis, we may assume that $\delta_{aa'} + \delta_{a'b} \geq 1$ is facet-defining for all instances on $n - 1 \geq 3$ jobs with $|\mathcal{P}(b')| \in \{0, 2\}$ for all $b' \in B$. Now, consider an instance on $n \geq 4$ jobs with $|\mathcal{P}(b')| \in \{0, 2\}$ for all $b' \in B$. We remove a job $j \in N \setminus \{a, a', b\}$ from the instance in such a way that we can apply the induction hypothesis. Then, we construct affinely independent feasible vectors based on the set of affinely independent vectors from the instance on $n - 1$ jobs. Feasibility of δ for Q then follows from feasibility of the constructed schedule.

Note that the removed job j can be chosen to have no successor. Either $B \setminus \{b\} \neq \emptyset$ or, if $B \setminus \{b\} = \emptyset$, then there is a job in $A \setminus \{a, a'\}$ without successors, since $|\mathcal{P}(b)| = 2$, see Figure 4.6 (middle and right). Hence, we can choose $j \in B \setminus \{b\}$, or $j \in A \setminus \{a, a'\}$ has neither predecessors nor successors. If we remove j (and all arcs that end in j) from the instance, we are left with an instance on $n - 1$ jobs. By our choice of j, this instance satisfies $|\mathcal{P}(b')| \in \{0, 2\}$ for all $b' \in B \setminus \{j\}$.

Thus, the induction hypothesis applies and there is a set D' of $d' = \frac{(n-1)(n-2)}{2}$ affinely independent vectors that satisfy $\delta_{aa'} + \delta_{a'b} = 1$. Note that these vectors correspond to feasible schedules on the instance without j. If we add j to the instance again, we add $n - 1$ variables, which we index by δ_{ij} for $i \in N \setminus \{j\}$.[29] Suppose the last $n - 1$ coordinates of the vectors correspond to these entries. We show how to extend the affinely independent vectors in D' to $d = d' + (n - 1)$ affinely independent vectors D in higher dimensional space. For the sake of simplicity, we omit "transpose" in the following and assume that all vectors are column vectors.

[29] Technically, we also add the variables δ_{ji} for all $i \in N \setminus \{j\}$ and δ_{jj}. However, δ_{jj} is set to 1 by (4.31e) and has no effect on the other constraints or (4.35). Also, δ_{ji} for $i \in N \setminus \{j\}$ is fixed by (4.31b) and does not influence (4.35). So, we can ignore these variables.

First, assume that j was chosen to have neither predecessors nor successors. Note that the vectors in $D_1 = \{(\delta, 0, \dots, 0) \mid \delta \in D'\}$ are feasible (j is scheduled first), affinely independent and satisfy (4.35) with equality. It holds $|D_1| = |D'| = d'$. Let $(\overline{\delta}, 0, \dots, 0) \in D_1$ be a schedule, where j starts first. We can successively move j "to the back" of this schedule without loosing feasibility. Thereby we obtain a set of vectors $D_2 = \{(\overline{\delta}, 1, 0, \dots, 0), (\overline{\delta}, 1, 1, 0, \dots, 0), \dots, (\overline{\delta}, 1, \dots, 1, 0), (\overline{\delta}, 1, \dots, 1)\}$ (up to permutation of the last $n-1$ coordinates). Note that the components that appear in (4.35) are not changed compared to $\overline{\delta}$, so all vectors in D_2 satisfy (4.35) with equality. Obviously, the set $D_1 \cup D_2$ is affinely independent and $|D_1 \cup D_2| = d' + n - 1 = d$, which proves the claim.

Now, assume that $j \in B \setminus \{b\}$. That is, it might not be feasible to schedule j first, and we cannot move j through the schedule as before. Consider a schedule $\overline{\delta}$ that schedules the jobs in order $a' \to b \to a \to (A \setminus \{a, a'\}) \to (B \setminus \{j, b\})$, where the sets $A \setminus \{a, a'\}$ and $B \setminus \{j, b\}$ are scheduled in any arbitrary order. Clearly, $\overline{\delta}$ is a feasible schedule for the instance on $n-1$ jobs and satisfies $\overline{\delta}_{aa'} + \overline{\delta}_{a'b} = 1$. Hence, we may, w.l.o.g., assume that $\overline{\delta} \in D'$. Let $\delta^j = (\overline{\delta}, 1, \dots, 1)$ be the schedule that first schedules $N \setminus \{j\}$ according to $\overline{\delta}$ and processes j last. Note that the vectors in $D_3 = \{(\delta, 1, \dots, 1) \mid \delta \in D'\} \ni \delta^j$ are feasible (j is scheduled last), affinely independent and satisfy (4.35) with equality by the induction hypothesis. It holds $|D_3| = |D'| = d'$. Next, we construct a set D_4 of $|D_4| = n - 1$ vectors such that the set $D_3 \cup D_4$ is affinely independent.

For every $i \in N \setminus \{j, a'\}$, let δ^i be the schedule that orders all jobs according to δ^j, but shifts i to the back of the schedule. That is, δ^i swaps the order of i and the set of jobs that appear after i in δ^j. In particular, $\delta^i_{ij} = 0$. Further, we define $\delta^{a'}$ to be the schedule that orders the jobs $a \to b \to (A \setminus \{a, a'\}) \to (B \setminus \{j, b\}) \to j \to a'$. So, compared to δ^j, job a' is moved to the back and the order of b and a is reversed (this is crucial to maintain feasibility). Let $D_4 = \{\delta^i \mid i \in N \setminus \{j\}\}$ with $|D_4| = n - 1$, and note that any $\delta^i \in D_4$ is feasible, since no job in B has exactly one predecessor.

For $i \neq a'$ we did not swap the order of a' and $\{b, a\}$ compared to $\overline{\delta}$, so δ^i satisfies (4.35) with equality. For $i = a'$, it holds $\delta^{a'}_{aa'} + \delta^{a'}_{a'b} = 1 + 0 = 1$. Further, for $i \in N \setminus \{j\}$, $\delta_{ij} = 1$ for all $\delta \in D_3$ and $\delta^k_{ij} = 0$ if and only if $k = i$ for all $\delta^k \in D_4$. So, $D_3 \cup D_4$ are $d' + n - 1 = d$ affinely independent feasible vectors that satisfy (4.35) with equality. This proves the claim. □

Recall that the fractional solution $\overline{\delta}$ in the proof of Lemma 4.12 satisfies (4.35), so the integrality gap remains linear, even with these additional constraints. In generalized min-sum set cover, each job b requires at least $\kappa(b) \in [|\mathcal{P}(b)|]$ of its predecessors to be completed before it can start. This can also be easily modeled with linear ordering variables by introducing a constraint $\sum_{a \in \mathcal{P}(b)} \delta_{ab} \geq \kappa(b)$. However, note that the instance in the proof of Lemma 4.12 is an instance of min-sum vertex cover (which is a special case of min-sum set cover and all-but-one min-sum set cover). So, already for $\kappa(b) = 1$ or $\kappa(b) = \max\{1, |\mathcal{P}(b)| - 1\}$ and $\Delta = 2$, the linear ordering formulation has an unbounded integrality gap.

4.4.2 Acyclic Precedence Graphs

In this section, we extend Theorem 4.13 from bipartite OR-scheduling to arbitrary acyclic precedence graphs. For this purpose, we need the notion of *bottleneck sets*. Informally, a job i is called a *bottleneck* if it is crucial for the execution of another job $j \neq i$. In this case, j is contained in the bottleneck set of i, i.e., the processing of j cannot start before i is completed.

Definition 4.15 (Bottlenecks and Bottleneck Sets)
Let N be a set of jobs and $G = (N, E_\vee)$ a precedence graph. We call $i \in N$ a *bottleneck* if i cannot be scheduled last in any feasible single-machine schedule. The set of bottlenecks is denoted by $\mathcal{B}$. For $i \in \mathcal{B}$, the *bottleneck set* of i is defined as $\mathcal{B}_i := \{j \in N \setminus \{i\} \mid j \text{ is preceded by } i \text{ in every feasible single-machine schedule}\}$. Further, we call $j \in \mathcal{B}_i$ *maximal in* $\mathcal{B}_i$ if there is no $k \in \mathcal{B}_i$ with $j \in \mathcal{B}_k$.

Note that $i \in \mathcal{B}$ implies $\mathcal{B}_i \neq \emptyset$. Let further $\overline{\mathcal{B}} := \bigcup_{i \in \mathcal{B}} \mathcal{B}_i$ be the set of all jobs that are contained in some bottleneck set. The next lemma summarizes some properties of bottleneck sets. In particular, the set of bottleneck sets $\{\mathcal{B}_i \mid i \in \mathcal{B}\}$ is laminar.

Lemma 4.16
Let $G = (N, E_\vee)$ be a precedence graph and let $i, k \in N$ and $k \in \mathcal{B}$. The following hold:

- *(i)* $i \in \mathcal{B}_k \implies \mathcal{B}_i \subseteq \mathcal{B}_k$
- *(ii)* $\mathcal{B}_i \cap \mathcal{B}_k \neq \emptyset$ *for* $i \neq k$ *implies* $i \in \mathcal{B}_k$ *or* $k \in \mathcal{B}_i$
- *(iii) If* $i \in \mathcal{B}$, *then* $i \in \mathcal{B}_k \iff \mathcal{B}_i \subsetneq \mathcal{B}_k$
- *(iv) For* $i \in \overline{\mathcal{B}}$, *there is exactly one* $k \in \mathcal{B}$ *such that* $i \in \mathcal{B}_k$ *is maximal in* $\mathcal{B}_k$
- *(v)* $\mathcal{B}_k = \bigcup \{\{i\} \dot\cup \mathcal{B}_i \mid i \text{ maximal in } \mathcal{B}_k\}$ *and this union is disjoint*

Proof. Recall the definition of the initial job j^{in} and the associated digraph G^{in} from Section 1.2.2. Note that $i \in \mathcal{B}_k$ if and only if every (j^{in}, i)-path in G^{in} contains k.

- *(i)* For $i \notin \mathcal{B}$, the claim is trivially true, as $\mathcal{B}_i = \emptyset \subseteq \mathcal{B}_k$, so assume $i \in \mathcal{B}$. For $j \in \mathcal{B}_i$, every (j^{in}, j)-path in G^{in} contains i and, since $i \in \mathcal{B}_k$, every (j^{in}, i)-path contains k. So, every (j^{in}, j)-path also contains k, which yields $j \in \mathcal{B}_k$.
- *(ii)* Let $i \neq k$ and $j \in \mathcal{B}_i \cap \mathcal{B}_k$, i.e., every (j^{in}, j)-path in G^{in} contains both i and k. Every such path contains i and k in a fixed order, say first k then i. Otherwise, if there was a (j^{in}, j)-path where i precedes k, then one could "shortcut" from i directly to j and "skip" k which contradicts $j \in \mathcal{B}_k$. This also implies that every (j^{in}, i)-path contains k, i.e., $i \in \mathcal{B}_k$. If the order of i and k in every (j^{in}, j)-path is reversed, then one obtains $k \in \mathcal{B}_i$ by similar arguments.
- *(iii)* Suppose $i \in \mathcal{B}$, i.e., $\mathcal{B}_i \neq \emptyset$. The first implication follows from *(i)* and the fact that $i \notin \mathcal{B}_i$, so $\{i\} \dot\cup \mathcal{B}_i \subseteq \mathcal{B}_k$. On the other hand, $\mathcal{B}_i \subsetneq \mathcal{B}_k$ implies $k \neq i$ and $\emptyset \neq \mathcal{B}_i = \mathcal{B}_i \cap \mathcal{B}_k$. The claim follows from *(ii)* and the fact that $k \notin \mathcal{B}_k \supsetneq \mathcal{B}_i$ implies $k \notin \mathcal{B}_i$.

(*iv*) If $i \in \mathcal{B}_k$ is not maximal then there exists $j \in \mathcal{B} \cap \mathcal{B}_k$ such that $i \in \mathcal{B}_j$. Further, $j \in \mathcal{B}_k$ implies $\mathcal{B}_j \subsetneq \mathcal{B}_k$ by (*iii*). Since $1 \leq |\mathcal{B}_j| < |\mathcal{B}_k|$, we can apply the same argument iteratively and obtain that i is maximal in some bottleneck set. Now suppose $i \in \overline{\mathcal{B}}$ is maximal in $\mathcal{B}_k$ and $\mathcal{B}_j$ for two jobs $k, j \in \mathcal{B}$ with $k \neq j$. In particular, $i \in \mathcal{B}_k \cap \mathcal{B}_j$. So, $k \in \mathcal{B}_j$ (or $j \in \mathcal{B}_k$) by (*ii*), which is a contradiction to the maximality of i in $\mathcal{B}_j$ (or $\mathcal{B}_k$).

(*v*) By (*i*), we can write $\mathcal{B}_k = \bigcup_{i \in \mathcal{B}_k} (\{i\} \cup \mathcal{B}_i)$. For every non-maximal element $j \in \mathcal{B}_k$, there exists $i \in \mathcal{B}_k$ with $\{j\} \cup \mathcal{B}_j \subseteq \mathcal{B}_i$. Iteratively, we obtain that every non-maximal element and its bottleneck set are contained in some $\mathcal{B}_i$ where $i \in \mathcal{B}_k$ is maximal in $\mathcal{B}_k$. By (*iv*), we can partition $\mathcal{B}_k$ into its maximal elements and their bottleneck sets. □

Note that we can compute bottlenecks and bottleneck sets in polynomial time by shortest path computations in G^{in}. Suppose we want to decide whether $i \in \mathcal{B}$ and, if so, return $\mathcal{B}_i$. We impose weights of zero on each arc, except for the incoming arcs into node i, to which we assign unit weight. Then, we compute all shortest (j^{in}, j)-paths in G^{in}, and all jobs j whose shortest path has non-zero length are contained in $\mathcal{B}_i$. Note that (*v*) in Lemma 4.16 is similar to (3.5).

Consider an instance of $1 \,|\, \textit{or-prec} \,|\, \sum w_j C_j$ with job set N' and an acyclic precedence graph $G' = (N', E')$. By introducing dummy jobs of zero processing time, we can transform the precedence graph such that every job has at most two predecessors. That is, for every $j \in N$ with $|\mathcal{P}(j)| \geq 3$, we replace the subgraph $G'[\{j\} \cup \mathcal{P}(j)]$ by a binary (in)tree with root j and one leaf for every $i \in \mathcal{P}(j)$. The "inner nodes" of this tree have zero processing time. Note that, if we do this for every job with more than two predecessors, we obtain a precedence graph $G = (N, E)$, where each job has at most two predecessors. Since the dummy jobs have zero processing time and the size of G is polynomial in the size of G', we can solve the initial OR-scheduling instance by solving the instance on G. Hence, assuming $|\mathcal{P}(j)| \leq 2$ for a job $j \in N$ is no restriction.

Recall the linear ordering LP (4.31) and the definition of bottleneck sets (Definition 4.15). Our main result in this section is an extension of Theorem 4.13.

Theorem 4.17
Let N be a set of jobs and $G = (N, E_\vee)$ an acyclic precedence graph. For all $j \in N$ and $\mathcal{P}(j) = \{i, i'\}$, the constraints

$$\delta_{ii'} + \delta_{i'j} \geq 1 \tag{4.37}$$

are valid for the integer hull of LP (4.31). Moreover, (4.37) holds with equality for all feasible integer solutions for LP (4.31) if $i \in \mathcal{B}_{i'}$, and it is facet-defining for the integer hull of LP (4.31) if $i \notin \mathcal{B}_{i'}$.

Note that validity of constraints (4.37) follows similarly as for (4.35) in the bipartite case. The case when i is part of the bottleneck set of i' is straight forward. We state this as a lemma to reference it later.

Lemma 4.18
Let $j \in N$ and $\mathcal{P}(j) = \{i, i'\}$. If $i \in \mathcal{B}_{i'}$, then $\delta_{ii'} + \delta_{i'j} = 1$ holds for all feasible integer solutions for LP (4.31).

Proof. Note that $i \in \mathcal{B}_{i'}$ and $\mathcal{P}(j) = \{i, i'\}$ imply $j \in \mathcal{B}_{i'}$. So, all feasible integer solutions δ for LP (4.31) satisfy $\delta_{i'j} = \delta_{i'i} = 1$, which together with (4.31b) proves the claim. □

The proof for the case $i \notin \mathcal{B}_{i'}$ of Theorem 4.17 is similar to the one of Lemma 4.14. The idea is, again, to remove one job and apply induction. However, the proof of Theorem 4.17 is more involved, since the removed job might alter some bottleneck sets. For instance, a job that was in no bottleneck set in G might become part of a bottleneck set in the smaller instance because we removed one of the "alternative paths" in G. Further, it is not clear rightaway whether the removed job can be chosen to have no predecessors or no successors, as in the proof of Lemma 4.14. The following lemma is an auxiliary result that allows us to restrict to cases where there exists a suitable job that can be removed.

Lemma 4.19
Let $G = (N, E_\vee)$ be an acyclic precedence graph. Let $j \in N$ with $\mathcal{P}(j) = \{i, i'\}$ and $n \geq 4$. If all jobs in $N \setminus \{i, i', j\}$ have predecessors and successors, i.e.,

$$\{k \in N \setminus \{i, i', j\} \mid \mathcal{P}(k) = \emptyset \text{ or } k \notin \mathcal{P}(k') \text{ for all } k' \in N\} = \emptyset, \tag{4.38}$$

then $\mathcal{B}_i = N \setminus \{i\}$ or $\mathcal{B}_{i'} = N \setminus \{i'\}$.

Proof. Suppose all jobs in $N \setminus \{i, i', j\}$ have predecessors and successors. So, all jobs in $N \setminus \{i, i', j\}$ are contained in (non-trivial) paths in G that begin and end in the set $\{i, i', j\}$. Since G is acyclic and j has only two predecessors, no such path begins or ends at j. Further, all paths go, w.l.o.g., from i to i', and there are no paths in the opposite direction. This implies that all jobs except for i itself are contained in the bottleneck set of i, i.e., $\mathcal{B}_i = N \setminus \{i\}$. Similarly, $\mathcal{B}_{i'} = N \setminus \{i'\}$ holds if all paths go from i' to i. □

Note that the bottleneck sets influence the dimension of the linear ordering polytope. Let $K = \sum_{k \in N} |\mathcal{B}_k|$ be the sum of the cardinalities of all bottleneck sets. Observe that, similar to [69, 140], the dimension of the integer hull of LP (4.31) is $d = \frac{n(n-1)}{2} - K$. The ratio $\frac{n(n-1)}{2}$ is due to the explicit equalities (4.31b) and (4.31e). Since $j' \in \mathcal{B}_k$ implies $\delta_{kj'} = 1$ by definition of a bottleneck set, we loose one dimension for every appearance of a job in another one's bottleneck set. The following lemma together with Lemma 4.18 concludes the proof of Theorem 4.17.

Fig. 4.7: Possible graphs G_1 (left) and G_2 (right) for the base case $n = 3$ and $i \notin \mathcal{B}_{i'}$.

Lemma 4.20
Let $G = (N, E_\vee)$ be an acyclic precedence graph. Let $j \in N$ and $\mathcal{P}(j) = \{i, i'\}$. If $i \notin \mathcal{B}_{i'}$, then constraint (4.37) is facet-defining for the integer hull of LP (4.31).

Proof. We prove the statement by exhibiting $d = \frac{n(n-1)}{2} - K$ affinely independent feasible integer solutions for LP (4.31) that satisfy $\delta_{ii'} + \delta_{i'j} = 1$. As before, we use δ interchangeably for schedules and the corresponding 0/1 vector, and omit "transpose" for the sake of simplicity and assume that all vectors are column vectors. The proof goes by induction on the number of jobs n.

The base case is $n = 3$, i.e., $d = 3 - K$. There are two possible acyclic precedence graphs, see Figure 4.7. For G_1, we have $K = 0$. Note that G_1 is the same as in the base case in the proof of Lemma 4.14, see also Figure 4.6 (left). For G_2, we have $K = 2$, since $\mathcal{B}_i = \{i', j\}$. The only feasible schedules for G_2 are $i \to i' \to j$ and $i \to j \to i'$. Clearly, only the second schedule satisfies (4.37) with equality. This establishes the base case for $n = 3$.

By induction hypothesis, we may assume that $\delta_{ii'} + \delta_{i'j} \geq 1$ is facet-defining for all instances on $n - 1 \geq 3$ jobs where i is not contained in the bottleneck set of i'. Recall that on instances where i is contained in the bottleneck set of i', the inequality is tight for all feasible integer vectors by Lemma 4.18. We remove a job $x \in N \setminus \{i, i', j\}$ from the instance and, depending on whether or not i is contained in the bottleneck set of i' on the smaller instance, we apply Lemma 4.18 or the induction hypothesis, respectively.

Let $x \in N \setminus \{i, i', j\}$ be a job with no predecessors or no successors. If no such job exists, i.e., all jobs in $N \setminus \{i, i', j\}$ have predecessors and successors, then Lemma 4.19 applies. The assumption $i \notin \mathcal{B}_{i'}$ and Lemma 4.19 yield $\mathcal{B}_i = N \setminus \{i\}$. In this case, let $x \in N \setminus \{i, i', j\}$ be maximal in $\mathcal{B}_i$ with $i \in \mathcal{P}(x)$.[30] Since $\mathcal{B}_i = N \setminus \{i\}$, job i is the only job without a predecessor and $\delta_{ik} = 1$ for all $k \in N \setminus \{i\}$ and all feasible schedules. That is, all feasible schedules process i first. Hence, we can remove i (and all components δ_{ik} and δ_{ki}) from the instance, adapt the parameters n and K accordingly, and treat x as if $\mathcal{P}(x) = \emptyset$.

If we remove x from the instance, we obtain an instance with $n - 1$ jobs. Let $\mathcal{B}'_k$ be the bottleneck set of $k \in N \setminus \{x\}$ on the instance with precedence graph $G[N \setminus \{x\}]$. It holds $\mathcal{B}'_k \supseteq \mathcal{B}_k \setminus \{x\}$ for all $k \in N \setminus \{x\}$, since x might have been a predecessor of some job in $\mathcal{B}'_k$ in the initial instance on G.

[30] Note that such a job exists, since $n \geq 4$ and all jobs other than i' and j lie on paths from i to i' in the precedence graph, see proof of Lemma 4.19.

Let $K' = \sum_{k \in N \setminus \{x\}} |\mathcal{B}'_k|$, and let D' be an inclusion-maximal affinely independent set of feasible integer solutions for LP (4.31) for the new instance on $G[N \setminus \{x\}]$ that satisfy (4.37) with equality. If $i \notin \mathcal{B}'_{i'}$, we get by induction hypothesis that

$$|D'| = d' := \frac{(n-1)(n-2)}{2} - K' = d - (n-1) + K - K'. \tag{4.39}$$

If $i \in \mathcal{B}'_{i'}$, then Lemma 4.18 implies $|D'| = d' + 1$. By adding x to the instance again, we add $n-1$ "free" variables, which we index by δ_{kx} for all $k \in N \setminus \{x\}$. Suppose the last $n-1$ coordinates of the vectors correspond to these entries. Similar to the proof of Lemma 4.14, we show how to extend the set D' to an affinely independent set of vectors in higher dimensional space. We distinguish several cases, depending on how the bottleneck sets in $G[N \setminus \{x\}]$ differ from those in the initial instance.

First, let us define a useful "standard schedule" $\overline{\delta}$ for the smaller instance on $G[N \setminus \{x\}]$. The standard schedule processes each bottleneck set contiguously, the jobs in $\mathcal{B}'_k \setminus \mathcal{B}_k$ are processed before those in $\mathcal{B}_k \setminus \{x\}$ for all $k \in N \setminus \{x\}$ (recall that $\mathcal{B}_k \setminus \{x\} \subseteq \mathcal{B}'_k$), and the jobs in $\mathcal{B}_x$ are scheduled last. Such a schedule exists and is feasible because the set of bottleneck sets $\{\mathcal{B}_k \mid k \in N\}$ is laminar (Lemma 4.16). Further, this schedule shall satisfy (4.37) with equality. Observe that $j \in \mathcal{B}_k$ for some $k \in N$ if and only if $\{i, i'\} \subseteq \{k\} \cup \mathcal{B}_k$, and similarly for $\mathcal{B}'_k$. So, $i' \in \mathcal{B}'_k \setminus \mathcal{B}_k$ and $i \in \{k\} \cup \mathcal{B}_k$ imply $j \in \mathcal{B}'_k \setminus \mathcal{B}_k$. For $i = k$, we can schedule the jobs in order $i \to j \to i'$ and, for $i \in \mathcal{B}_k$, we can schedule $i' \to j \to i$. In either case, constraint (4.37) is satisfied with equality. Similarly, if i and i' are interchanged. Hence, there is a feasible schedule $\overline{\delta}$ with the following properties:

- (i) the jobs in $\mathcal{B}'_k$ are scheduled contiguously directly after k for all $k \in N \setminus \{x\}$,
- (ii) the jobs in $\mathcal{B}_k \setminus \{x\}$ are scheduled after those in $\mathcal{B}'_k \setminus \mathcal{B}_k$ for all $k \in N \setminus \{x\}$,
- (iii) the jobs in $N \setminus (\{x\} \cup \mathcal{B}_x)$ precede those in $\mathcal{B}_x$, and
- (iv) j is scheduled directly before the latter of the jobs i and i'.

Note that condition (iv) implies $\overline{\delta}_{ii'} + \overline{\delta}_{i'j} = 1$. We have to distinguish two main cases, namely whether x has successors or not.

If x has no successors, then $\mathcal{B}'_k = \mathcal{B}_k \setminus \{x\}$ and $\mathcal{B}_x = \emptyset$. The set of vectors $D_0 = \{(\delta, 1, \dots, 1) \mid \delta \in D'\}$ is feasible (x is scheduled last) and affinely independent. Let $S = \{k \in N \setminus \{x\} \mid x \notin \mathcal{B}_k\}$ be the set of jobs that do not contain x in their bottleneck set, and let $p = |S|$. Note that $K - K' = n - 1 - p$. We construct a set D_1 of cardinality $p = d - d'$ vectors such that $D_0 \cup D_1$ is affinely independent.

Note that $\mathcal{B}_k = \mathcal{B}'_k$ for all $k \in S$. For any $k \in S$, we consider a feasible schedule δ^k for the instance $G[N \setminus \{x\}]$ that processes the jobs in $\{k\} \cup \mathcal{B}_k$ last. For $k \in S \setminus \{i, i'\}$, we can take $\overline{\delta}$ and shift $\{k\} \cup \mathcal{B}_k$ to the back of the schedule. If $i' \in S$, we might have to reverse the order of i and j to maintain feasibility. If i' precedes j in $\overline{\delta}$, then j is scheduled directly before i in $\overline{\delta}$ by (iv). Hence, we can swap the order of i and j, and move i' and its bottleneck set to the back. The resulting schedule $\delta^{i'}$ is feasible.

That is, $\delta^{i'}$ orders the jobs $i \to j \to i' \to \mathcal{B}_{i'}$ (with other jobs before i' according to $\overline{\delta}$). If $i \in S$ and $i' \notin \mathcal{B}_i$, then $j \notin \mathcal{B}_i$. So, similar to $\delta^{i'}$, the vector δ^i can be chosen to process the jobs $i' \to j \to i \to \mathcal{B}_i$ (with other jobs before i according to $\overline{\delta}$). If $i \in S$ and $i' \in \mathcal{B}_i$, we can choose δ^i such that j is the first job of $\mathcal{B}_i$, i.e., $i \to j \to (\mathcal{B}_i \setminus \{j\})$ (with other jobs before i according to $\overline{\delta}$). In either case, $\delta^k_{ii'} + \delta^k_{i'j} = 1$ for $k \in S$, i.e., (4.37) is satisfied with equality.

For every $k \in S$, consider the schedule δ^k and process x directly before k, which is feasible, since x has no successors and by the definition of S. Suppose the last $n-1$ coordinates corresponding to δ_{kx} (with $k \in N \setminus \{x\}$) are of the block structure $(\delta_{\overline{S}}, \delta_S) \in \{0,1\}^{n-1-p} \times \{0,1\}^p$, where $\delta_{\overline{S}}$ and δ_S correspond to all pairs (k,x) with $k \notin S$ and $k \in S$, respectively. Let $D_1 = \{(\delta^k, \delta^k_{\overline{S}}, y^k) \,|\, k \in S\}$, where $y^k \in \{0,1\}^{|S|}$ with $y^k_{k'x} = 0$ if and only if $k' \in (\{k\} \cup \mathcal{B}_k) \cap S$, be the set of vectors corresponding to these schedules. It holds $|D_1| = |S| = p$ and the set D_1 is affinely independent.

To see this, recall that the set of bottleneck sets is laminar. If $\mathcal{B}_k \cap \mathcal{B}_{k'} = \emptyset$ for $k, k' \in S$, then the vectors y^k and $y^{k'}$ have zeros at different coordinates. If the bottleneck sets intersect, then, w.l.o.g., $\{k'\} \cup \mathcal{B}_{k'} \subseteq \mathcal{B}_k$. In this case, $y^k_{kx} = 0 \neq y^{k'}_{kx}$. Further, $D_0 \cup D_1$ is affinely independent, since every vector in D_1 has at least one zero-entry in its last $|S|$ components, whereas the vectors in D_0 have all ones. It holds $|D_0 \cup D_1| = d' + p = d' + n - 1 - K + K' = d$, see (4.39). This proves the statement for the case that x has no successors.

If x has successors then $\mathcal{P}(x) = \emptyset$ by our choice of x. When we add x to the instance, $|\mathcal{B}_x|$ jobs become part of its bottleneck set. On the other hand, adding x might also lessen some bottleneck sets, so $K \leq K' + |\mathcal{B}_x|$. Recall the "standard schedule" $\overline{\delta}$ and consider the set of vectors $D_2 = \{(\delta, 0, \ldots, 0) \,|\, \delta \in D'\}$. The vectors in D_2 are feasible (x is scheduled first), affinely independent and $|D_2| \in \{d', d'+1\}$ depending on whether $i \notin \mathcal{B}'_{i'}$ (induction hypothesis) or $i \in \mathcal{B}'_{i'}$ (Lemma 4.18). We construct two sets D_3 and D_4 such that $D = D_2 \cup D_3 \cup D_4$ are d affinely independent vectors.

As for D_3, consider the schedule $(\overline{\delta}, 0, \ldots, 0)$ that schedules the jobs according to $\overline{\delta}$ and processes x first. Recall that $\overline{\delta}$ processes the jobs in $\mathcal{B}_x$ last by *(iii)*. Note that $\mathcal{B}_x$ might be empty, i.e., $|\mathcal{B}_x| = 0$. Suppose the last $|\mathcal{B}_x|$ coordinates correspond to the job pairs (k,x) with $k \in \mathcal{B}_x$. These entries form a block of zeros, which we denote by $\delta_{\mathcal{B}_x}$. Similar to the proof of Lemma 4.14, we can successively move x directly before the first job of $\mathcal{B}_x$ without loosing feasibility. Thereby, we obtain a set of vectors

$$D_3 = \{(\overline{\delta}, y, \delta_{\mathcal{B}_x}) \,|\, y \in \{(1,0,\ldots,0), \ldots, (1,\ldots,1)\} \subseteq \{0,1\}^{n-1-|\mathcal{B}_x|}\}. \tag{4.40}$$

The coordinates in y correspond to the pairs (k,x) with $k \notin \mathcal{B}_x$. Note that $D_2 \cup D_3$ are affinely independent, all vectors in D_3 satisfy (4.37) with equality and $|D_3| = n-1-|\mathcal{B}_x|$.

If $K = K' + |\mathcal{B}_x|$ then $\mathcal{B}_k = \mathcal{B}'_k$ for all $k \in N \setminus \{x\}$, i.e., x does not lessen any bottleneck sets. In particular, $|D_2| = d'$ by induction hypothesis ($i \notin \mathcal{B}_{i'} = \mathcal{B}'_{i'}$) and $|D_2 \cup D_3| = d' + n - 1 - |\mathcal{B}_x| = d - K' + K - |\mathcal{B}_x| = d$, see (4.39), yields the claim.

So assume $K < K' + |\mathcal{B}_x|$, i.e., there are $k \in N \setminus \{x\}$ with $\mathcal{B}'_k \supsetneq \mathcal{B}_k$. Note that

$$\begin{aligned} K' + |\mathcal{B}_x| - K &= \sum_{k \in N \setminus \{x\}} |\mathcal{B}'_k| + |\mathcal{B}_x| - \sum_{k \in N} |\mathcal{B}_k| \\ &= \sum_{k \in N \setminus \{x\}} (|\mathcal{B}'_k| - |\mathcal{B}_k|) = \sum_{k \in N \setminus \{x\}} |\mathcal{B}'_k \setminus \mathcal{B}_k|. \end{aligned} \tag{4.41}$$

As for the construction of D_4, we exhibit affinely independent sets D_k for all $k \in N \setminus \{x\}$ with $\mathcal{B}'_k \setminus \mathcal{B}_k \neq \emptyset$ so that $D_4 = \bigcup_k D_k$. Consider again the schedule $(\overline{\delta}, 0, \dots, 0)$ that processes x first. Recall that $\overline{\delta}$ schedules k directly before $\mathcal{B}'_k$ by *(i)*, and $\mathcal{B}'_k \setminus \mathcal{B}_k$ before $\mathcal{B}_k$ by *(ii)*. For $k \notin \{i, i'\}$, let D_k be the set of vectors obtained from the schedule $(\overline{\delta}, 0, \dots, 0)$ when shifting k successively before the first job of $\mathcal{B}_k$. Similar to (4.40), we obtain $|\mathcal{B}'_k \setminus \mathcal{B}_k|$ affinely independent vectors.

For $k \in \{i, i'\}$ this argument also works if $k' \in \{i, i'\} \setminus \{k\}$ is not contained in $\mathcal{B}'_k \setminus \mathcal{B}_k$. Recall that $\overline{\delta}$ orders the jobs $k \to j \to k'$ (with some other jobs between k and j). If $k' \in \mathcal{B}'_k \setminus \mathcal{B}_k$, then $j \in \mathcal{B}'_k \setminus \mathcal{B}_k$, and we have to be careful when moving k "to the back", similar to the construction of, e.g., $\delta^{i'}$ before. That is, we move k directly before j, reorder the jobs $\{k, j, k'\}$ in a feasible way such that (4.37) is satisfied with equality, and then move k further until it is processed directly before the first job of $\mathcal{B}_k$.

More precisely, suppose $k' \in \mathcal{B}'_k \setminus \mathcal{B}_k$. For $k = i$, we have $k' = i'$. In this case, when we reach a schedule where $i \to j \to i'$ are scheduled contiguously, which occurs by *(iv)*, we reorder the jobs to $i' \to i \to j$ (while keeping all other jobs unchanged), and then continue with moving i further to the back. This gives in total $|D_i| = |\mathcal{B}'_i \setminus \mathcal{B}_i|$ affinely independent vectors. For $k = i'$, we get $k' = i$. When we reach the contiguous ordering $i' \to j \to i$, we reorder the jobs such that $i \to j \to i'$ (while keeping all other jobs unchanged).[31] Thereby, we obtain $|D_{i'}| = |\mathcal{B}'_{i'} \setminus \mathcal{B}_{i'}| - 1$ affinely independent vectors. Note that this only occurs if $i \in \mathcal{B}'_{i'} \setminus \mathcal{B}_{i'}$. If $i \notin \mathcal{B}'_{i'}$, then $|D_{i'}| = |\mathcal{B}'_{i'} \setminus \mathcal{B}_{i'}|$.

Now, set $D_4 = \bigcup_k D_k$ and $D = D_2 \cup D_3 \cup D_4$. Similar to y in (4.40), the entries $(\delta_{k'_1 k}, \dots, \delta_{k'_\ell k})$ for $k'_q \in \mathcal{B}'_k \setminus \mathcal{B}_k$ form a sequence of vectors $(1, 0, \dots, 0), \dots, (1, \dots, 1, 0)$, $(1, \dots, 1) \in \{0,1\}^{|\mathcal{B}'_k \setminus \mathcal{B}_k|}$ when shifting k directly before $\mathcal{B}_k$. Since all entries $\delta_{k'k}$ are equal to zero for $k' \in \mathcal{B}'_k \setminus \mathcal{B}_k$ and $\delta \in D \setminus D_k$, we get that D is an affinely independent set. As for the cardinality of D, observe that, if $i \in \mathcal{B}'_{i'} \setminus \mathcal{B}_{i'}$, it holds $|D_2| = d' + 1$ by Lemma 4.18 and $D_{i'} = |\mathcal{B}'_{i'} \setminus \mathcal{B}_{i'}| - 1$. Else, $|D_2| = d'$ by induction hypothesis and $|D_k| = |\mathcal{B}'_k \setminus \mathcal{B}_k|$ for all $k \in N \setminus \{x\}$. In either case,

$$\begin{aligned} |D| &= |D_2| + |D_3| + |D_4| = d' + n - 1 - |\mathcal{B}_x| + \sum_{k \in N \setminus \{x\}} |\mathcal{B}'_k \setminus \mathcal{B}_k| \\ &= d' + n - 1 - |\mathcal{B}_x| + K' + |\mathcal{B}_x| - K = d, \end{aligned} \tag{4.42}$$

where the third and last equality is due to (4.41) and (4.39), respectively. This proves the claim. □

[31] We have to exclude the order $i \to i' \to j$, which would be similar to the case $k = i$, because the inequality (4.37) is strict for this ordering.

4.5 Completion Time Formulation

In this section, we extend the parallel inequalities of [167, 139] and present a completion time formulation for OR-scheduling in Section 4.5.2. We then show that this LP has an unbounded gap between optimal LP solution and optimal feasible schedule, even on bipartite instances with $\Delta = 2$. We start out with a necessary notion in Section 4.5.1.

4.5.1 Generalized Minimal Chains

The following definition extends the notion of minimal chains from Definition 2.5. Recall that, in Definition 2.5, a minimal chain of a job $k \in N$ is a set of jobs that needs to be scheduled to complete k as early as possible. Now, consider an arbitrary subset of jobs $S \subseteq N$. We define the length of a minimal chain w.r.t. the set S to be the amount of extra time we need at least to complete k, provided that the jobs in S are already fixed in the schedule. The minimal chain of k w.r.t. S is the set of jobs in $N \setminus S$ that have to be processed to complete k in that time.

Definition 4.21 (Generalized Minimal Chains)
Let N be a set of jobs and $G = (N, E_\vee)$ a precedence graph. Let $\mathcal{FS} \subseteq 2^N$ be the set of feasible starting sets. For an arbitrary subset $S \subseteq N$ and a job $k \in N$, we define the *length of the minimal chain of k w.r.t. S* as

$$mc(S,k) \coloneqq \min\left\{ \sum_{j\in L} p_j \,\middle|\, L \subseteq N : \exists\, \widehat{S} \subseteq S \cup L \text{ with } k \in \widehat{S} \in \mathcal{FS} \right\}. \tag{4.43}$$

A set $L \in \operatorname{argmin}\{mc(S,k)\}$ is called a *minimal chain* of k w.r.t. S. The set of minimal chains is denoted by $\mathcal{MC}(S,k)$.

In the absence of release dates, $\mathcal{MC}(\emptyset,k) = \mathcal{MC}(k)$, i.e., Definition 4.21 generalizes Definition 2.5. The function $mc(\cdot,k)$ is non-increasing, i.e., $mc(S,k) \geq mc(S',k)$ for all $k \in N$ and $S \subseteq S' \subseteq N$. Since the processing times are non-negative, there exists a minimal chain $L \in \mathcal{MC}(S,k)$ with $L \cap S = \emptyset$. If we enumerate the jobs of a minimal chain $L = \{j_1, \dots, j_\ell\} \in \mathcal{MC}(S,k)$ suitably, then $\{j_1, \dots, j_q\} \in \mathcal{MC}(S, j_q)$ for all $q \in [\ell]$. In other words, if job j is contained in a minimal chain L, then the jobs preceding j in L form a minimal chain of j.

Observe that the value $mc(S,k)$ equals the length of a shortest path from the initial job j^{in} to k in G^{in} if we impose weights of 0 and p_j on all arcs (i,j) for $j \in S$ and $j \notin S$, respectively. We denote this weighted directed graph by $G^{\text{in}}(S)$. Recall that shortest paths can be computed in polynomial time, see, e.g., [124, 102]. That is, given a set $S \subseteq N$ and a job $k \in N$, we can compute a minimal chain $L \in \mathcal{MC}(S,k)$ in polynomial time. Figure 4.8 illustrates the minimal chains of a job k w.r.t. two different sets in the corresponding weighted digraphs.

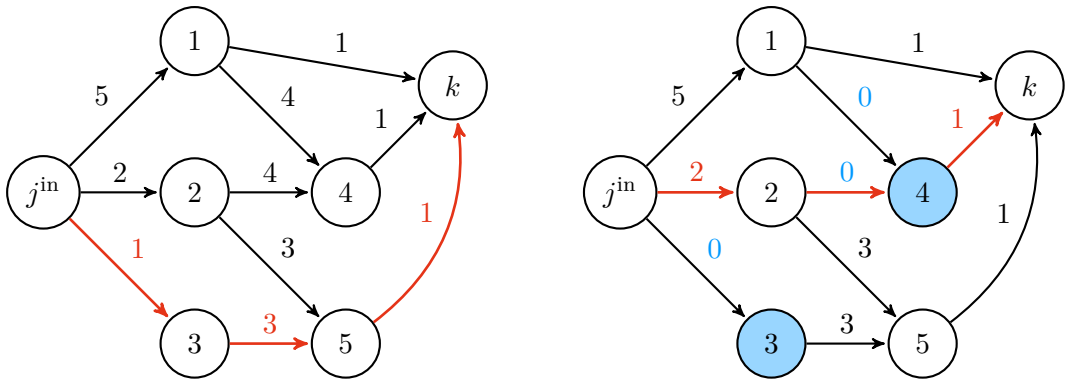

Fig. 4.8: Weighted digraphs $G^{\text{in}}(\emptyset)$ (left) and $G^{\text{in}}(\{3,4\})$ (right) for an instance with processing times $p_1 = 5$, $p_2 = 2$, $p_3 = p_k = 1$, $p_4 = 4$ and $p_5 = 3$. The red arcs correspond to the paths of the minimal chains $\{3,5,k\} \in \mathcal{MC}(\emptyset, k)$ and $\{2,k\} \in \mathcal{MC}(\{3,4\}, k)$ of k w.r.t. the empty set and $\{3,4\}$, respectively.

Before we proceed with the LP formulation, we observe some properties of the set function $mc(\cdot, k) : 2^N \to \mathbb{N}_0$. Two important notions for set functions are submodularity and supermodularity, which refine subadditivity and superadditivity, respectively. A set function $g : 2^N \to \mathbb{Z}$ is called *submodular* if $g(S \cup \{j\}) - g(S) \geq g(S' \cup \{j\}) - g(S')$ for all $S \subseteq S' \subseteq N$ and $j \notin S'$. A function g is called *supermodular* if $-g$ is submodular. We call g *modular* if it is both submodular and supermodular. The following lemma summarizes the inclusion-minimal and inclusion-maximal graph classes where $mc(\cdot, k)$ is (not) submodular and (not) supermodular (anymore), respectively.

Lemma 4.22
Let N be a set of jobs, $k \in N$ and $G = (N, E_\vee)$ be a precedence graph. The minimal chain function $mc(\cdot, k) : 2^N \to \mathbb{N}_0$ is

- *(i) modular if G is an outforest,*
- *(ii) supermodular but not submodular if G is bipartite, and*
- *(iii) neither supermodular nor submodular if G is an intree (but not a chain).*

Proof. Throughout the proof, let $S \subseteq S' \subseteq N$ and $j \in N \setminus S'$.

(i) Let G be an outforest, and let $\widehat{S} \subseteq N$ be the set of jobs on the unique path that starts at a root of G and ends at k. Since G is an outforest, we have $\widehat{S} \setminus S \in \mathcal{MC}(S, k)$ and $\widehat{S} \setminus S' \in \mathcal{MC}(S', k)$. It holds

$$\begin{aligned} mc(S \cup \{j\}, k) - mc(S, k) &= p\left(\widehat{S} \setminus (S \cup \{j\})\right) - p\left(\widehat{S} \setminus S\right) \\ &= \begin{cases} 0 & \text{if } j \notin \widehat{S}, \\ -p_j & \text{if } j \in \widehat{S}. \end{cases} \end{aligned} \tag{4.44}$$

Similarly for S'. So, $mc(S \cup \{j\}, k) - mc(S, k) = mc(S' \cup \{j\}, k) - mc(S', k)$.

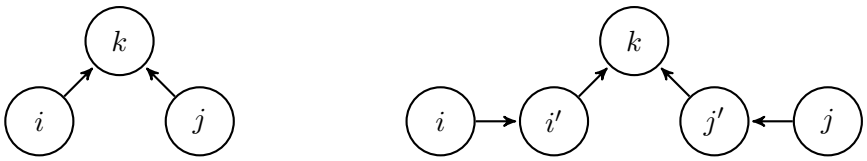

Fig. 4.9: Precedence graphs for which $mc(\cdot, k)$ is not submodular (left) and not supermodular (right). All processing times are equal to 1. For the left instance, choose $S = \emptyset$ and $S' = \{i\}$, and for the right instance, let $S = \{i\}$ and $S' = \{i, j'\}$.

(*ii*) Let $G = (A \dot\cup B, E_\vee)$ with $E_\vee \subseteq A \times B$ be bipartite. To prove that $mc(\cdot, k)$ is supermodular, we need to distinguish several cases. First suppose that $k \in A$. Then, $mc(S, k) = p_k$ if $k \notin S$ and $mc(S, k) = 0$ otherwise. For $j = k$, we get $mc(S \cup \{k\}, k) = 0 = mc(S' \cup \{k\}, k)$, and $mc(S, k) = p_k = mc(S', k)$, since $k \notin S' \supseteq S$. If $j \neq k$, then $mc(S \cup \{j\}, k) = mc(S, k)$ and $mc(S' \cup \{j\}, k) = mc(S', k)$, since j does not contribute to a minimal chain of k. (Recall that there are no ingoing arcs at $k \in A$ because $E_\vee \subseteq A \times B$.) So, in either case, $mc(S \cup \{j\}, k) - mc(S, k) = mc(S' \cup \{j\}, k) - mc(S', k)$.

Now suppose that $k \in B$, and let $i \in \operatorname{argmin}\{p_{i'} \mid i' \in \mathcal{P}(k)\}$ be a potential candidate in a minimal chain of k. (W.l.o.g., we can assume $p_i > 0$.) That is, $mc(S, k) = p_i + p_k$ if $S \cap (\mathcal{P}(k) \cup \{k\}) = \emptyset$, $mc(S, k) = p_i$ if $S \cap (\mathcal{P}(k) \cup \{k\}) = \{k\}$, $mc(S, k) = p_k$ if $S \cap \mathcal{P}(k) \neq \emptyset$ and $k \notin S$, and $mc(S, k) = 0$ if $S \cap \mathcal{P}(k) \neq \emptyset$ and $k \in S$. Similarly for S'. For $j = k$, we get that $mc(S \cup \{k\}, k) - mc(S, k) = -p_k = mc(S' \cup \{k\}, k) - mc(S', k)$, since $k \notin S' \supseteq S$ and the term p_i (if it appears) cancels out. If $j \neq k$ and $j \in B$, then including j into the sets has no effect on the length of a minimal chain, i.e., $mc(S \cup \{j\}, k) = mc(S, k)$ and $mc(S' \cup \{j\}, k) = mc(S', k)$. In any case, $mc(S \cup \{j\}, k) - mc(S, k) = mc(S' \cup \{j\}, k) - mc(S', k)$.

Finally, suppose that $k \in B$, $j \in A$ and $i \in \operatorname{argmin}\{p_{i'} \mid i' \in \mathcal{P}(k)\}$ with $p_i > 0$. If $j \notin \mathcal{P}(k)$, then including j to any of the sets has no effect, i.e., $mc(S \cup \{j\}, k) = mc(S, k)$ and $mc(S' \cup \{j\}, k) = mc(S', k)$. For $j \in \mathcal{P}(k)$, it holds $mc(S \cup \{j\}, k) - mc(S, k) \in \{-p_i, 0\}$, depending on whether or not $S \cap \mathcal{P}(k) = \emptyset$. Similarly for S'. If $mc(S \cup \{j\}, k) - mc(S, k) = -p_i$, then $mc(S \cup \{j\}, k) - mc(S, k) \leq mc(S' \cup \{j\}, k) - mc(S', k)$ holds in any case. Further, $mc(S \cup \{j\}, k) - mc(S, k) = 0$ implies that adding j to S has no effect on the length of a minimal chain. So S already contains a job in $\mathcal{P}(k)$, i.e., $S \cap \mathcal{P}(k) \neq \emptyset$. Since $S \subseteq S'$ we get that also $S' \cap \mathcal{P}(k) \neq \emptyset$, and thus $mc(S' \cup \{j\}, k) - mc(S', k) = 0$. On the other hand, $mc(S' \cup \{j\}, k) - mc(S', k) = -p_i$ implies $S' \cap \mathcal{P}(k) = \emptyset$ because adding j to S' affects the length of a minimal chain. Since $S' \supseteq S$, we get $S \cap \mathcal{P}(k) = \emptyset$, so $mc(S \cup \{j\}, k) - mc(S, k) = -p_i$. Hence, $mc(\cdot, k)$ is supermodular if G is bipartite.

To see that $mc(\cdot, k)$ is not necessarily submodular, consider the instance in Figure 4.9 (left), which is bipartite. For $S = \emptyset$ and $S' = \{i\}$, we obtain

$$mc(S \cup \{j\}, k) - mc(S, k) = -1 < 0 = mc(S' \cup \{j\}, k) - mc(S', k). \tag{4.45}$$

So, $mc(\cdot, k)$ is not submodular.

(iii) The instances depicted in Figure 4.9 are both intrees. We already observed that $mc(\cdot, k)$ is not necessarily submodular, see (4.45) and Figure 4.9 (left). For the instance on the right of Figure 4.9, set $S = \{i\}$ and $S' = \{i, j'\}$. Then

$$mc(S \cup \{j\}, k) - mc(S, k) = 0 > -1 = mc(S' \cup \{j\}, k) - mc(S', k). \tag{4.46}$$

So, the minimal chain function $mc(\cdot, k)$ is neither submodular nor supermodular for intrees that are no chains. □

Recall from Figure 1.1 that Lemma 4.22 indeed covers all graph classes from Section 1.1.2. In particular, the minimal chain function is neither supermodular nor submodular for inforests or general acyclic graphs, and it is supermodular and submodular for chains and outtrees.

4.5.2 The Minimal Chain Relaxation

Recall the parallel inequalities (Proposition 4.1) of Wolsey [167] and Queyranne [139] and the definition of generalized minimal chains (Definition 4.21). For $k \in N$, we define a function $g_k : 2^N \to \mathbb{N}_0$ via

$$g_k(S) \coloneqq \frac{1}{2}\Big(\sum_{j \in S} p_j + mc(S, k)\Big)^2 + \frac{1}{2}\Big(\sum_{j \in S} p_j^2 + mc(S, k)^2\Big) \qquad \text{for all } S \subseteq N. \tag{4.47}$$

Note that $g_k(S) = g(S)$ if $k \in S$ and $S \in \mathcal{FS}$ is a feasible starting set, since then $mc(S, k) = 0$. That is, g_k extends the function g from Section 4.2.1. We can generalize the parallel inequalities (4.1) to what we call *minimal chain inequalities*. The next theorem is similar to Proposition 4.1. However, in contrast to Proposition 4.1, we only get a necessary condition for the feasibility of completion time vectors.

Theorem 4.23
Let N be a set of jobs and $G = (N, E_\vee)$ a precedence graph. Any completion time vector $C \in \mathbb{R}^N$ that corresponds to a feasible single-machine schedule w.r.t. G satisfies the minimal chain inequalities

$$\sum_{j \in S} p_j C_j + mc(S, k)\, C_k \geq g_k(S) \qquad \textit{for all } k \in N \textit{ and } S \subseteq N. \tag{4.48}$$

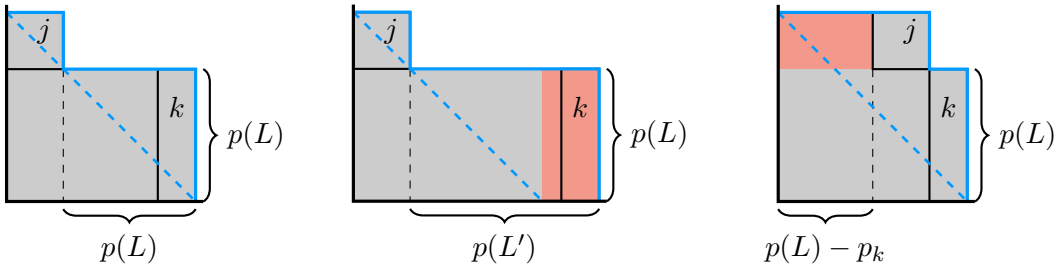

Fig. 4.10: Excerpt of the two-dimensional Gantt chart corresponding to the optimal schedule with $j \in S$ (left). Note that there are rectangles of jobs in $L \setminus \{k\}$ that have zero height between the rectangles corresponding to j and k. If we replace $L \in \mathcal{MC}(S,k)$ by some L' such that $k \in S \cup L' \in \mathcal{FS}$, we obtain the picture in the middle with $p(L') \geq p(L)$. The difference in the area is highlighted in red. The right picture depicts the situation that a job $j \in S$ is scheduled between the first job of L and k.

Proof. For $k \notin S$, it holds $mc(S,k) = p_k + mc(S \cup \{k\}, k)$, and thus $g_k(S) \geq g_k(S \cup \{k\})$. Note that the left hand sides of (4.48) for S and $S \cup \{k\}$ coincide in this case. So, for $k \in S$, (4.48) is dominated by the corresponding inequality for k and $S \setminus \{k\}$. Further, for $k \in S \in \mathcal{FS}$, it holds $mc(S,k) = 0$, so (4.48) is equivalent to the parallel inequality of [167, 139]:

$$\sum_{j \in S} p_j C_j \geq g_k(S) = \frac{1}{2}\Big(\sum_{j \in S} p_j\Big)^2 + \frac{1}{2}\Big(\sum_{j \in S} p_j^2\Big) = g(S). \tag{4.49}$$

Note that all unit vectors have positive scalar product with the left hand side of (4.48), so idle time in a schedule only increases the left hand side of (4.48). Hence, it suffices to show that all completion time vectors of schedules without idle time satisfy (4.48) for $k \in N$ and $S \subseteq N \setminus \{k\}$ with $S \notin \mathcal{FS}$.

Fix $k \in N$ and $S \subseteq N \setminus \{k\}$ such that $S \notin \mathcal{FS}$ is not a feasible starting set. Let $L \in \mathcal{MC}(S,k)$ be a minimal chain and note that $k \in L$. We define weights $w_j = p_j$ for $j \in S$, $w_k = mc(S,k)$ and $w_j = 0$ for $j \in N \setminus (S \cup \{k\})$. Consider a schedule (which is not necessarily feasible) that processes all jobs in S before those in L, and all jobs in $N \setminus (S \cup L)$ last. Further, k shall be the last job in L. Similar to Figure 4.2 in Section 4.2.1, one can verify that the objective value of such a schedule is equal to $g_k(S)$. Figure 4.10 (left) illustrates an excerpt of the two-dimensional Gantt chart.

We now argue why any feasible schedule has an objective value of at least $g_k(S)$. Clearly, processing jobs in $N \setminus (S \cup L)$ before k increases the objective function value. By the definition of $L \in \mathcal{MC}(S,k)$, altering L, i.e., exchanging some jobs in L, only increases the width of the rectangle corresponding to L. That is, among all feasible schedules, the contribution of k to the area of the Gantt chart is minimal if k is preceded by the jobs in $L \setminus \{k\}$, see Figure 4.10 (middle).

By Smith's rule [157], the objective value of the schedule is independent of the particular order of the jobs in $S \cup \{L\}$, where we interpret L as a single job (all "jobs" in $S \cup \{L\}$ have Smith ratio equal to 1). Suppose we want to schedule a job $j \in S$ between the first job of L and k. By Smith's rule, we can assume that j is the job that directly precedes the first job of L in the schedule. Then, by moving j between the first job of L and k, the completion time of j is increased, whereas the completion time of k remains, see Figure 4.10 (right). Thus, the objective function value increases. Hence, no feasible schedule has lower objective value than the schedule that processes the jobs in the order $S \to (L \setminus \{k\}) \to k \to (N \setminus (S \cup L))$. Recall that this schedule has objective value equal to $g_k(S)$, so any feasible schedule satisfies (4.48). □

Note that inequality (4.48) is satisfied with equality for a feasible schedule that processes the jobs in $S \cup L$ before those in $N \setminus (S \cup L)$ and schedules the jobs in L contiguously with k being the last job in L. Theorem 4.23 suggests the following natural *minimal chain relaxation* for $1 \,|\, or\text{-}prec \,|\, \sum w_j C_j$:

$$\min \quad \sum_{j \in N} w_j \, C_j \tag{4.50a}$$

$$\text{s.t.} \quad \sum_{j \in S} p_j \, C_j + mc(S,k) \, C_k \geq g_k(S) \qquad \forall \, k \in N, \ \forall \, S \subseteq N, \tag{4.50b}$$

$$C_j \geq 0 \qquad \forall \, j \in N. \tag{4.50c}$$

The parallel inequalities are implicitly contained in (4.50b) for $k \in S \in \mathcal{FS}$. Note that it is not clear how to separate constraints (4.50b) in polynomial time. That is, it is not clear whether LP (4.50) can be solved in polynomial time, which is in contrast to the corresponding completion time LP (4.4) for AND-scheduling [139, 148, 71]. Even if we could solve LP (4.50) in polynomial time, scheduling the jobs in non-decreasing order of their LP values, similar to [148], does not necessarily yield a feasible schedule.

Let $\ell \in \mathbb{N}$. Consider a bipartite OR-scheduling instance with $n = 2\ell + 1$ jobs and sets $A = \{a, i_1, \ldots, i_\ell\}$ and $B = \{j_1, \ldots, j_\ell\}$. The processing times and weights are $p_a = \frac{\ell}{2}$, $w_a = 0$, and $p_{i_q} = w_{j_q} = 1$, $w_{i_q} = p_{j_q} = 0$ for all $q \in [\ell]$. The predecessors of jobs in B are $\mathcal{P}(j_q) = \{a, i_q\}$ for all $q \in [\ell]$. Figure 4.11 on the next page illustrates the precedence graph. One can check that the vector $\overline{C}$ defined as $\overline{C}_{j_q} = 1$, $\overline{C}_{i_q} = q + 1$ for all $q \in [\ell]$ and $\overline{C}_a = \frac{3}{2}\ell + 1$ is feasible for LP (4.50), see proof of Lemma 4.24. Constraints (4.50b) with $k = j_q$ and $S = \emptyset$ imply $C_{j_q} \geq mc(\emptyset, j_q) = 1$ for all $q \in [\ell]$. Since the jobs in $\{j_1, \ldots, j_\ell\}$ are the only ones with non-zero weight, $\overline{C}$ is an optimal solution for LP (4.50). It satisfies $\overline{C}_{j_q} < \overline{C}_{i_q} < \overline{C}_a$ for all $q \in [\ell]$. Thus, scheduling the jobs in non-decreasing order of their $\overline{C}_j$ values does not yield a feasible solution.

The above instance also shows that the gap of LP (4.50) can grow linearly in the number of jobs, even for instances with bipartite precedence graphs and $\Delta = 2$. Recall that, for AND-scheduling, the gap between an optimal LP solution of LP (4.4) and an optimal feasible schedule is equal to 2 [148, 71].

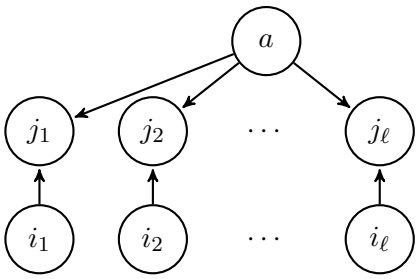

Fig. 4.11: Instance that rules out a list scheduling algorithm with the LP values of an optimum solution of LP (4.50) and for which LP (4.50) exhibits an "integrality gap" that is linear in the number of jobs. The processing times and weights are $p_a = \frac{\ell}{2}$, $w_a = w_{i_q} = p_{j_q} = 0$, and $w_{j_q} = p_{i_q} = 1$ for all $q \in [\ell]$.

Lemma 4.24
There is a family of instances such that the gap between an optimal solution of LP (4.50) and an optimal schedule is $\Omega(n)$.

Proof. Consider the instance in Figure 4.11 described above. It holds $mc(\emptyset, i_q) = mc(\emptyset, j_q) = 1 < \frac{\ell}{2} = mc(\emptyset, a)$ for all $q \in [\ell]$. Note that $\Delta = 2$. Due to the structure of the precedence relation, there are only two reasonable ways to schedule the jobs that are feasible. Let C' and C'' be the completion time vectors of schedules that order the jobs $a \to \{j_1, \ldots, j_\ell\} \to \{i_1, \ldots, i_\ell\}$ and $\{i_q \to j_q\} \to a$, respectively. The notion $\{i_q \to j_q\}$ indicates that we schedule pairs $i_q \to j_q$ for all $q \in [\ell]$ consecutively in arbitrary order. The objective function values of C' and C'' are equal to $\frac{\ell^2}{2}$ and $\sum_{q=1}^{\ell} q = \frac{\ell(\ell+1)}{2}$, respectively. One can easily verify that any other schedule, which is not of a similar form as C' and C'', has a strictly larger objective value than C' or C''. Since $\frac{\ell^2}{2} < \frac{\ell(\ell+1)}{2}$, C' is an optimal schedule with an objective value of $\Omega(n^2)$.

Recall the solution $\overline{C}$ defined as $\overline{C}_{j_q} = 1$, $\overline{C}_{i_q} = q + 1$ for all $q \in [\ell]$ and $\overline{C}_a = \frac{3}{2}\ell + 1$. The objective function value of $\overline{C}$ is equal to $\ell \in \mathcal{O}(n)$, so the gap of the objective function values of C' and $\overline{C}$ is $\Omega(n)$. It remains to be shown that $\overline{C}$ is feasible for LP (4.50), i.e., it satisfies constraints (4.50b). Note that $\overline{C}$ corresponds to a schedule with idle time on the single machine (i.e., no jobs overlap) of the following form: $idle \to \{j_1, \ldots, j_\ell\} \to \{i_1, \ldots, i_\ell\} \to idle \to a$. That is, it satisfies the parallel inequalities $\sum_{j \in S} p_j \overline{C}_j \geq g(S)$ for all $S \subseteq N$ of [167, 139]. We show that $\overline{C}$ also satisfies constraints (4.50b).

Let $k \in A$ and $S \subseteq N$. The value of $mc(S, k)$ is equal to 0 (if $k \in S$) or p_k (if $k \notin S$). In either case, we obtain

$$g_k(S) = \frac{1}{2}\Big(\sum_{j \in S \cup \{k\}} p_j\Big)^2 + \frac{1}{2}\Big(\sum_{j \in S \cup \{k\}} p_j^2\Big) = g(S \cup \{k\}), \tag{4.51}$$

and, thus,

$$\sum_{j\in S} p_j\,\overline{C}_j + mc(S,k)\,\overline{C}_k = \sum_{j\in S\cup\{k\}} p_j\,\overline{C}_j \geq g(S\cup\{k\}) = g_k(S). \tag{4.52}$$

Recall that $p_j = 0$ for $j \in B$, i.e., all terms in (4.50b) for $j \in B \cap S$ are zero. So, it suffices to restrict to $S \subseteq A$ in the following. Let $k = j_q \in B$ for some $q \in [\ell]$, and set $h = |S \cap \{i_1, \ldots, i_\ell\}| \leq \ell$. First, suppose $a \in S$, so $mc(S,k) = 0$. It holds

$$g_k(S) = \frac{1}{2}\left(\frac{\ell}{2} + h\right)^2 + \frac{1}{2}\left(\frac{\ell^2}{4} + h\right) = \frac{\ell^2}{4} + \frac{1}{2}\,h^2 + \frac{\ell+1}{2}\,h. \tag{4.53}$$

We obtain

$$\begin{aligned}
&\sum_{j\in S} p_j\,\overline{C}_j + mc(S,k)\,\overline{C}_k = p_a\,\overline{C}_a + \sum_{i\in S\cap\{i_1,\ldots,i_\ell\}} p_i\,\overline{C}_i \\
&\geq \frac{\ell}{2}\left(\frac{3}{2}\ell + 1\right) + \sum_{q=1}^{h}(q+1) \geq \frac{\ell^2}{4} + \left(\frac{\ell}{2}+1\right)h + \frac{h(h+1)}{2} \\
&\geq \frac{\ell^2}{4} + \frac{\ell+1}{2}\,h + \frac{1}{2}h^2 = g_k(S).
\end{aligned} \tag{4.54}$$

The first inequality holds with equality if the h jobs in $S \cap \{i_1, \ldots, i_\ell\}$ are those with lowest indices, otherwise the inequality is strict. For the second inequality, we use $\ell + 1 \geq h$.

If $a \notin S$, and $i_q \in S$, we get $mc(S,k) = 0$ and $g_k(S) = \frac{1}{2}\,h^2 + \frac{1}{2}\,h = \frac{h(h+1)}{2}$. Similar to before, we obtain

$$\sum_{j\in S} p_j\,\overline{C}_j + mc(S,k)\,\overline{C}_k = \sum_{i\in S\cap\{i_1,\ldots,i_\ell\}} p_i\,\overline{C}_i \geq \sum_{q=1}^{h}(q+1) \geq \frac{h(h+1)}{2} = g_k(S). \tag{4.55}$$

Finally, if $\mathcal{P}(k) \cap S = \emptyset$, then $mc(S,k) = 1$. So,

$$g_k(S) = \frac{1}{2}(h+1)^2 + \frac{1}{2}(h+1) = \frac{h+1}{2}\,(h+2), \tag{4.56}$$

and

$$\begin{aligned}
\sum_{j\in S} p_j\,\overline{C}_j + mc(S,k)\,\overline{C}_k &= \sum_{i\in S\cap\{i_1,\ldots,i_\ell\}} p_i\,\overline{C}_i + \overline{C}_k \geq \sum_{q=1}^{h}(q+1) + 1 \\
&= \frac{h(h+1)}{2} + h + 1 = \frac{h+1}{2}\,(h+2) = g_k(S).
\end{aligned} \tag{4.57}$$

Hence, $\overline{C}$ satisfies constraints (4.50b) for all $k \in N$ and $S \subseteq N$, and is feasible for LP (4.50). □

Lemma 4.24 shows that one needs further (stronger) valid inequalities than the minimal chain inequalities (4.50b) to obtain an LP based constant-factor approximation algorithm similar to, e.g., [148], for (bipartite) OR-scheduling.

4.6 Open Problems

Lemmas 4.12 and 4.24 indicate that the linear ordering formulation and completion time formulation fail to derive LP based approximation algorithms for OR-scheduling problems. Therefore, we base our algorithms in Section 4.3 on time-indexed linear programs. The approximation ratios of 2Δ for bipartite AND/OR-scheduling (Theorem 4.4) and 4 for all-but-one min-sum set cover (Theorem 4.10) yield upper bounds on the integrality gaps of LPs (4.14) and (4.27), respectively. For LP (4.27), Lemma 4.11 gives a matching lower bound of 4 and shows that the analysis of the 4-approximation algorithm is tight. It is not clear whether the analysis of Algorithm 5 in Section 4.3.1 is best possible. In view of the integrality gaps in Sections 4.4 and 4.5, it would be interesting to obtain stronger bounds on the integrality gap of LP (4.14).

Problem 4.25
What is the integrality gap of the time-indexed LP (4.14)?

Erlebach, Kääb and Möhring [43] presented a lower bound of $2^{(\log n)^{1-\gamma}}$ with $\gamma = (\log\log n)^{-c}$ and $0 < c < \frac{1}{2}$ on the approximability for AND/OR-scheduling. The set of jobs in the reduction of [64, 96] from Label Cover consists of five disjoint sets $N_1 \dot\cup N_2 \dot\cup N_3 \dot\cup N_4 \dot\cup N_5$ and the precedence graph is of a layered OR/AND/OR/AND structure, i.e., $E_\wedge \subseteq (N_2 \times N_3) \cup (N_4 \times N_5)$ and $E_\vee \subseteq (N_1 \times N_2) \cup (N_3 \times N_4)$. Further, the graph restricted to the set $N \setminus N_1$ is an intree, see [64, 96]. Although this graph already seems very restricted, the precedence graph for a bipartite AND/OR-scheduling instance is not of this structure. Hence, there might be a better (or even constant-factor) approximation algorithm for this case. Recall that the reduction from Hitting Set to bipartite AND/OR-scheduling shows that there is no constant-factor approximation if $E_\wedge \cap (B \times B) \neq \emptyset$ in general, unless P = NP. Note that the precedence relation $E_\wedge \cap (B \times B)$ in the reduction is a bipartite intree, i.e., already very restrictive. However, constant-factor approximation algorithms are within the realms of possibility if, for instance, $E_\wedge \subseteq A \times A$.

Problem 4.26
Are there constant-factor approximation algorithms for restricted cases of bipartite AND/OR-scheduling?

Answering this question in the affirmative could also give an answer to the planted dense subgraph conjecture [24, 122]. Recall that, if the conjecture is true, obtaining a constant-factor approximation for precedence-constrained min-sum set cover is NP-hard [122]. Hence, if we had a constant-factor approximation algorithm for bipartite AND/OR-scheduling with $E_\wedge \subseteq A \times A$, this would disprove the planted dense subgraph conjecture.

Problem 4.27
Prove or disprove the planted dense subgraph conjecture.

Since the approximation factor for bipartite AND/OR-scheduling only depends on the parameter Δ, we obtain a 4-approximation for precedence-constrained min-sum vertex cover (Corollary 4.7). The unconstrained version of min-sum vertex cover can be approximated within a factor of 16/9 [13] and is APX-hard [45]. That is, there is an $\varepsilon > 0$ such that it is NP-hard to approximate MSVC better than $1 + \varepsilon$. Natural open questions are to improve the 16/9-approximation of [13], to improve the lower bound on the approximation factor, or to improve the approximation factor of 4 for precedence-constrained MSVC.

Problem 4.28
Close the gap of $[1 + \varepsilon; 4]$ in the approximation guarantee of precedence-constrained min-sum vertex cover.

In Section 4.3.2, we present a tight 4-approximation algorithm for all-but-one min-sum set cover. Feige, Lovász and Tetali [45] showed that 4 is best possible for min-sum set cover, unless P = NP. This gives a lower bound of 4 on the approximability of generalized min-sum set cover. For the special case of all-but-one min-sum set cover, the only known lower bound is APX-hardness from min-sum vertex cover [45].

Problem 4.29
Is there a (combinatorial) $(4 - \varepsilon)$-approximation algorithm for all-but-one min-sum set cover?

Even though the LP formulations in linear ordering and completion time variables seem to fail in the presence of OR-precedence constraints (Lemmas 4.12 and 4.24), one could try to find stronger valid inequalities for either of these formulations that cut off the presented counterexamples. To solve the completion time formulation LP (4.50) in polynomial time, we also require a polynomial-time separation oracle for the minimal chain inequalities (4.50b).

Problem 4.30
Can constraints (4.50b) be separated in polynomial time?

Chapter 5

Preemptive Concurrent Open Shop with Release Dates

In this last chapter, we consider a scheduling environment that is different from the previously considered problems. We introduce the setting and give some related work in Section 5.1, and then devise a 2-approximation for the preemptive variant in Section 5.2.

5.1 Introduction and Related Work

The problem considered here is the preemptive variant of *concurrent open shop with release dates*. Let $m, n \in \mathbb{N}$. We are given a set M of m machines and a set N of n jobs. Each job $j \in N$ consists of m operations $\{O_{ij} \mid i \in M\}$, one for each machine. Operation O_{ij} requires $p_{ij} \geq 0$ units of time on machine $i \in M$ and is released at time $r_{ij} \geq 0$. An operation may have zero processing time, but we assume there is at least one operation of each job and on each machine that has non-zero processing time. Otherwise, we may disregard this job or machine, respectively. Finally, each job $j \in N$ is assigned a non-negative weight $w_j \geq 0$. Similar to Section 1.1.4, we can assume that all data are integral by scaling suitably.

The *completion time* of operation O_{ij} is denoted by C_{ij}, and it is defined as the first point in time at which its processing is completed. The completion time of job $j \in N$ is then defined as the completion time of its latest operation, i.e., $C_j \coloneqq \max_{i \in M} C_{ij}$.

Definition 5.1 (Preemptive Concurrent Open Shop with Release Dates) Let M be a set of machines and N a set of jobs with operations $\{O_{ij} \mid i \in M\}$ for each $j \in N$. A schedule is called *feasible* if

(i) operation O_{ij} is processed for p_{ij} units of time by machine i,
(ii) no operation O_{ij} starts before its release date r_{ij}, and
(iii) every machine processes at most one operation at a time.

Note that operations may be preempted. The task is to find a feasible schedule that minimizes the sum of weighted completion times, $\sum_{j \in N} w_j C_j$.

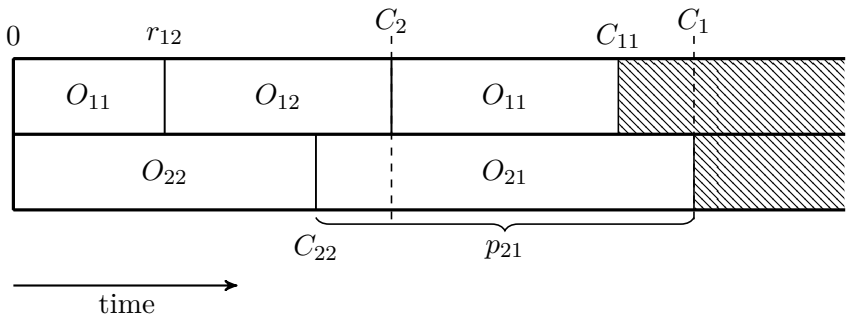

Fig. 5.1: A feasible preemptive schedule for two jobs and two machines. Operation O_{11} is preempted so that O_{12} can start at its release date (all other release dates are equal to zero). The lengths of the operations indicate the processing times.

Following the notation of [116], we denote this problem by $PD \,|\, r_{ij},\, pmtn \,|\, \sum w_j C_j$ in an extension of the notation of [68]. Figure 5.1 illustrates a feasible schedule for preemptive concurrent open shop with release dates for two jobs and two machines. If $m = 1$ or if all operations of a job have equal processing times ($p_{ij} = p_j$) and equal release dates ($r_{ij} = r_j$), then the problem is equivalent to the single-machine problem $1 \,|\, r_j,\, pmtn \,|\, \sum w_j C_j$, which is already strongly NP-hard [106]. If all operations of a job have equal release dates, i.e. $r_{ij} = r_j$, we call the release dates *job-dependent*. In the related *open shop* problem, no two operations of the same job may be scheduled simultaneously. This is obviously more restrictive, and the schedule in Figure 5.1 would not be feasible for open shop. We refer to [111, 27, 136, 114] for more details.

Related Work. Concurrent open shop scheduling was first studied by Ahmadi and Bagchi [4], and is known to be strongly NP-hard, even if $m = 2$ and all release dates are trivial [168]. In the absence of release dates, Wang and Chen [163] presented an approximation algorithm with a guarantee of $\frac{16}{3}$ based on an interval-indexed linear program. Several research groups decreased the approximation factor to 2 using LP relaxations in completion time variables [29, 55, 117, 121]. The algorithms in [29, 55, 117] solve the LPs exactly, whereas Mastrolilli et al. [121] presented a combinatorial Greedy algorithm and used the LP only for the analysis. Using a construction of [121], Bansal and Khot [16] and Kumar et al. [105] independently showed that 2 is essentially best possible under a variant of the Unique Games Conjecture [100]. Sachdeva and Saket [143] strengthened this inapproximability result and proved that there is no $(2 - \varepsilon)$-approximation, unless P = NP. Note that there is no benefit in preemption if all release dates are trivial. Thus, the 2-approximations and inapproximability results carry over to the preemptive variant without release dates.

To the best of the author's knowledge, concurrent open shop has only been considered with job-dependent release dates, $r_{ij} = r_j$, so far and not with general release dates r_{ij}. Non-preemptive concurrent open shop with job-dependent release dates can be approximated within a factor of 3 [55, 117, 5]. The algorithms of Garg, Kumar and Pandit [55] and Leung, Li and Pinedo [117] both solve an LP relaxation in completion time variables and then schedule the operations in non-decreasing order of their LP solution values. Ahmadi et al. [5] consider a more general problem of scheduling coflows and propose a combinatorial 3-approximation algorithm that is analyzed via an LP in completion time variables and that generalizes the 2-approximation of [121]. Im et al. [87] discuss matroid coflows and, as a by-product, present a $(2+\varepsilon)$-approximation for preemptive concurrent open shop scheduling with job-dependent release dates. Their algorithm is based on a time-indexed linear program.

5.2 A 2-Approximation Algorithm

We observe that a standard linear programming relaxation together with an algorithm of Hall et al. [71] for the related single-machine problem, $1 \,|\, r_j,\, pmtn \,|\, \sum w_j C_j$, already achieves an approximation guarantee of 2 for $PD \,|\, r_{ij},\, pmtn \,|\, \sum w_j C_j$. In contrast to previous work [55, 117, 5, 87], we consider a more general problem, where each operation O_{ij} has an individual release date r_{ij}. Our approach avoids the additive $\varepsilon > 0$ in the approximation guarantee of [87], and, thus, we obtain the best possible approximation factor for this problem (unless P = NP, see [143]).

We introduce the LP relaxation of the problem in Section 5.2.1. Then, we show that scheduling the operations on each machine in non-decreasing order of their LP solution according to a simple list scheduling algorithm of [71] gives a 2-approximate solution in Section 5.2.2.

5.2.1 A Valid LP Relaxation

The LP relaxation is based on completion time variables introduced by Wolsey [167] and Queyranne [139], see Section 4.2.1. The relaxation presented here generalizes the ones of [141, 60, 61, 62] for the single-machine problem and of Mastrolilli et al. [121] for concurrent open shop scheduling without release dates. Let C_{ij} be a variable that indicates the completion time of operation O_{ij} in a schedule. Note that for every $i \in M$, the schedule restricted to the operations $\{O_{ij} \,|\, j \in N\}$ is a feasible single-machine schedule. Recall from Proposition 4.1 that any schedule satisfies the constraints

$$\sum_{j \in S} p_{ij}\, C_{ij} \geq \frac{1}{2}\Big(\sum_{j \in S} p_{ij}\Big)^2 + \frac{1}{2}\sum_{j \in S} p_{ij}^2 \qquad \forall\, S \subseteq N,\ \forall\, i \in M. \tag{5.1}$$

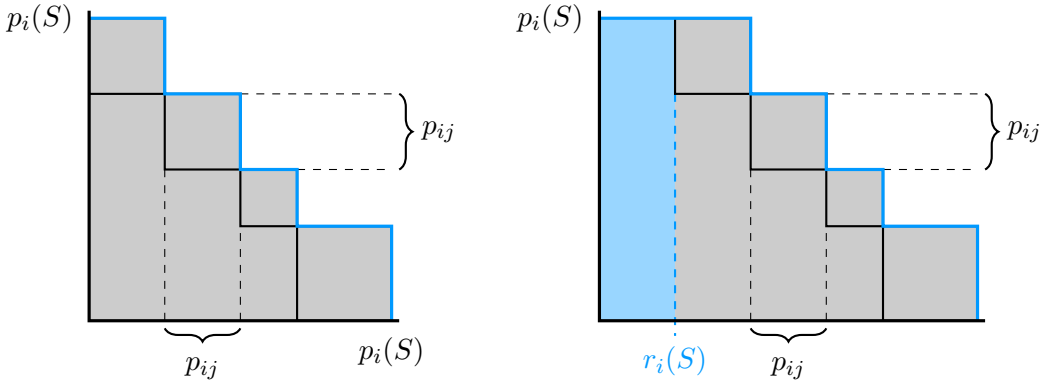

Fig. 5.2: Geometric interpretation of (5.1) (left) and the corresponding shifted Gantt chart for constraints (5.2) (right). The gray area on the left underneath the upper envelope of the rectangles (blue line) equals the right hand side of (5.1). The area on the right corresponds to the right hand side of (5.2). It equals the blue area, which is precisely $r_i(S)\,p_i(S)$, plus the gray area on the left.

For $S \subseteq N$ and $i \in M$, let $p_i(S) \coloneqq \sum_{j\in S} p_{ij}$ and $r_i(S) \coloneqq \min_{j\in S} r_{ij}$. Let $S \subseteq N$ such that $r_i(S) > 0$. Recall the interpretation of (5.1) via a two-dimensional Gantt chart, where the area of the Gantt chart equals the right hand side of (5.1), see Figure 4.2 in Section 4.2.1 and [42, 63]. No operation in $\{O_{ij} \mid j \in S\}$ can start before time $r_i(S)$. So, compared to (5.1), the completion time of each operation has to be increased by at least $r_i(S)$ to obtain a feasible schedule. That is, the Gantt chart is "shifted to the right" by an amount of $r_i(S)$, see Figure 5.2. Thus, we obtain the following valid inequality for all $S \subseteq N$ and $i \in M$, see also [141, 60, 61, 62]:

$$\sum_{j\in S} p_{ij}\,C_{ij} \geq \frac{1}{2}\Bigg(\sum_{j\in S} p_{ij}\Bigg)^2 + \frac{1}{2}\sum_{j\in S} p_{ij}^2 + r_i(S)\,p_i(S). \tag{5.2}$$

Using $\frac{1}{2}\sum_{j\in S} p_{ij}^2 \geq 0$, we relax (5.2) and obtain the valid inequalities

$$\sum_{j\in S} p_{ij}\,C_{ij} \geq p_i(S)\Big(r_i(S) + \frac{1}{2}\,p_i(S)\Big) \qquad \forall S \subseteq N,\ \forall\, i \in M. \tag{5.3}$$

Thereby, we obtain the following LP relaxation for $PD\,|\,r_{ij}, pmtn\,|\,\sum w_jC_j$ with variables C_j and C_{ij} for $j \in N$ and $i \in M$:

$$\text{min} \qquad \sum_{j\in N} w_j\,C_j \tag{5.4a}$$

$$\text{s.t.} \qquad \sum_{j\in S} p_{ij}\,C_{ij} \geq p_i(S)\left(r_i(S) + \frac{1}{2}\,p_i(S)\right) \qquad \forall i \in M,\, \forall\, S \subseteq N, \tag{5.4b}$$

$$C_j \geq C_{ij} \qquad \forall i \in M,\, \forall\, j \in N. \tag{5.4c}$$

Although the number of constraints is exponential, we can solve LP (5.4) in polynomial time by efficiently separating constraints (5.4b), see [141, 60]. For any machine i, the schedule of operations $\{O_{ij} \mid j \in N\}$ is a feasible single-machine schedule, which gives constraints (5.4b). Further, constraints (5.4c) link the completion times of the operations to the completion time of the respective job and, thus, to the objective function. So, the optimal objective value $\sum_{j \in N} w_j \overline{C}_j$ of LP (5.4) is a lower bound on the optimal objective value of a feasible preemptive schedule. We can relax LP (5.4) further by removing the variables C_{ij}, dropping constraints (5.4c) and replacing (5.4b) accordingly:

$$\min \quad \sum_{j \in N} w_j C_j \tag{5.5a}$$

$$\text{s.t.} \quad \sum_{j \in S} p_{ij} C_j \geq p_i(S) \left(r_i(S) + \frac{1}{2} p_i(S) \right) \qquad \forall i \in M, \forall S \subseteq N. \tag{5.5b}$$

Note that if $p_{ij} \neq 0$, we can rewrite constraints (5.5b) for $S = \{j\}$ and $i \in M$ as $C_j \geq r_{ij} + \frac{1}{2} p_{ij} \geq 0$. That is, the completion time of each job is greater or equal than the release dates of its operations. We assume that, for $S = \{j\}$ and $i \in M$ with $p_{ij} = 0$, constraints (5.5b) are simplified to $C_j \geq r_{ij}$, although we do not state this explicitly in LP (5.5).

5.2.2 Preemptive List Scheduling

In this section, we use an optimal solution of LP (5.5) to construct a preemptive feasible schedule. To this end, let $\overline{C}_j$ be the completion time corresponding to a fixed optimal solution of LP (5.5) of job $j \in N$. We call $\overline{C}_j$ the *fractional completion time* of j. First, we sort the jobs in non-decreasing order of their fractional completion times $\overline{C}_{\sigma^{-1}(1)} \leq \overline{C}_{\sigma^{-1}(2)} \leq \cdots \leq \overline{C}_{\sigma^{-1}(n)}$, where $\sigma : N \to [n]$ is a linear ordering of N, and ties are broken arbitrarily. Then, we schedule the i-th operations of the jobs on machine i according to the preemptive list scheduling algorithm of Hall et al. [71], which works as follows.

Fix $i \in M$ and consider the i-th operations of the jobs in order of the list σ one at a time. For operation O_{ij}, let $t \geq r_{ij}$ be the first point in time after r_{ij} when machine i is idle. Let $t' > t$ be maximal such that machine i is idle in the interval $[t; t']$. If $t' - t \geq p_{ij}$, then operation O_{ij} can be scheduled completely within this interval, starting at time t. Otherwise, start operation O_{ij} at time t and schedule it for $t' - t$ time units. Then, preempt O_{ij} at time t', and continue it at the next point in time when the machine is idle. Once machine i processed operation O_{ij} for p_{ij} time units, pick the next operation O_{ik} with $\sigma(k) = \sigma(j) + 1$ in the list. Repeat the above procedure until all operations $\{O_{ij} \mid j \in N\}$ are scheduled. The complete algorithm is summarized in Algorithm 6.

Input: Instance of $PD \,|\, r_{ij},\, pmtn \,|\, \sum w_j C_j$
Output: A feasible schedule for $PD \,|\, r_{ij},\, pmtn \,|\, \sum w_j C_j$

```
1  Solve LP (5.5) and let C̄ be an optimal solution;
2  Let σ : N → [n] so that C̄_{σ⁻¹(1)} ≤ ··· ≤ C̄_{σ⁻¹(n)} (break ties arbitrarily);
3  for i ∈ M do
4      for ℓ = 1, ..., n do
5          Let j ∈ N such that σ(j) = ℓ, set t' ← r_ij and p ← p_ij;
6          while p > 0 do
7              t ← min{s ≥ t' | machine i is idle at time s};
8              t' ← max{s > t | machine i is idle in [t; s]};
9              if t' − t ≥ p then
10                 schedule operation O_ij in [t; t + p] on machine i, set p ← 0;
11             else
12                 schedule operation O_ij partially in [t; t'] on machine i;
13                 set p ← p − (t' − t);
14             end
15         end
16     end
17 end
18 return schedule;
```

Algorithm 6: A 2-approximation algorithm for $PD \,|\, r_{ij},\, pmtn \,|\, \sum w_j C_j$.

Note that if operation O_{ij} has to be preempted at time t', then another operation O_{ik} with $\sigma(k) < \sigma(j)$ is released at time t'. Let C_{ij} be the completion time of operation O_{ij} in the schedule returned by Algorithm 6. Clearly, we get a feasible preemptive single-machine schedule on every machine $i \in M$. The following lemma is similar to Lemma 2.8 in [71]. We include the proof for completeness.

Lemma 5.2
The completion times of the schedule returned by Algorithm 6 satisfy $C_{ij} \leq 2\,\overline{C}_j$ for all $j \in N$ and all $i \in M$.

Proof. Fix $i \in M$, and rename the jobs such that $\overline{C}_1 \leq \overline{C}_2 \leq \cdots \leq \overline{C}_n$. Let $j \in N$ and consider the iteration of the algorithm where operation O_{ij} was placed on machine i. That is, consider a partial schedule where only jobs in $[j] = \{1, \ldots, j\}$ are placed on the machine.

Let S_{ik} be the starting time of operation O_{ik}. Let $S \subseteq \{k \in [j] \,|\, C_{ik} \leq C_{ij}\}$ be the set of jobs such that there is no idle time in the interval $[S_{ik}; C_{ij}]$ in the partial schedule. Note that the set S is not empty, since $j \in S$. Let $j' \in S$ be the job with $r_{ij'} = r_i(S)$.

There is no idle time in $[r_{ij'}; C_{ij'}]$ because we could have scheduled (parts of) operation $O_{ij'}$ earlier otherwise. So, in the interval $[r_i(S); C_{ij}]$, machine i is busy with jobs in S and all jobs in S are completed. Thus,

$$C_{ij} = r_i(S) + \sum_{k \in S} p_{ik} \leq 2 \left(r_i(S) + \frac{1}{2} p_i(S) \right). \tag{5.6}$$

If $p_i(S) = 0$, then $C_{ij} = r_{ij} \leq \overline{C}_j \leq 2\,\overline{C}_j$ by constraints (5.5b). Else,

$$p_i(S)\, C_{ij} \leq 2\, p_i(S) \left(r_i(S) + \frac{1}{2}\, p_i(S) \right) \leq 2 \sum_{k \in S} p_{ik}\, \overline{C}_k \leq 2\, p_i(S)\, \overline{C}_j. \tag{5.7}$$

The first and second inequality are due to (5.6) and (5.5b), respectively. The last inequality holds, since $\overline{C}_k \leq \overline{C}_j$ for all $k \in S \subseteq [j]$. The claim now follows if we divide both sides of (5.7) by $p_i(S) > 0$. □

In summary, we obtain the following main result of this chapter.

Theorem 5.3
Algorithm 6 is a 2-approximation algorithm for $PD \,|\, r_{ij},\, pmtn \,|\, \sum w_j C_j$.

Proof. Recall that we can solve LP (5.5) and construct the schedules on each machine $i \in M$ in polynomial time. That is, Algorithm 6 runs in polynomial time. If we process the operations on machine i according to Algorithm 6, we obviously obtain a feasible schedule for $PD \,|\, r_{ij},\, pmtn \,|\, \sum w_j C_j$. The completion time of job j is $C_j = \max_{i \in M} C_{ij}$ and Lemma 5.2 yields

$$\sum_{j \in N} w_j\, C_j = \sum_{j \in N} w_j \max_{i \in M} C_{ij} \leq \sum_{j \in N} w_j \max_{i \in M} 2\, \overline{C}_j = 2 \sum_{j \in N} w_j\, \overline{C}_j. \tag{5.8}$$

This proves the claim. □

Recall that we can assume that all data are integral, and observe that there is no need to preempt if $p_{ij} \in \{0, 1\}$ for all $i \in M$ and $j \in N$. So, Theorem 5.3 immediately yields the following corollary.

Corollary 5.4
Algorithm 6 is a 2-approximation algorithm for $PD \,|\, r_{ij},\, p_{ij} \in \{0, 1\} \,|\, \sum w_j C_j$.

5.3 Open Problems

Due to a result of [143], the approximation ratio of Theorem 5.3 is tight, unless P = NP. However, Algorithm 6 requires the "heavy machinery" of separating constraints (5.5b).

The $(2+\varepsilon)$-approximation algorithm of Im et al. [87] does not need separation, but is based on a time-indexed LP the size of which is exponential in the input. When decreasing the size of this LP to be polynomial in the input, the approximation guarantee increases by $\varepsilon > 0$, similar to our results in Section 4.3. It would be an interesting research question to design a purely combinatorial algorithm similar to the algorithms for concurrent open shop without release dates [121] and coflows with release dates [5].

Problem 5.5
Is there a combinatorial 2-approximation algorithm for $PD \,|\, r_{ij},\, pmtn \,|\, \sum w_j C_j$?

It seems natural that applying a non-preemptive list scheduling algorithm instead of the algorithm of [71] in Algorithm 6 should give a 2-approximation for $PD \,|\, r_{ij} \,|\, \sum w_j C_j$. So far, the best approximation ratio for non-preemptive concurrent open shop with job-dependent release dates is 3 [55, 117, 5]. All these algorithms use LPs in completion time variables with constraints $C_{ij} \geq r_{ij} + p_{ij}$ and (5.1), although in [5] it is only used for the analysis of a combinatorial algorithm, similar to [121]. One possibility to obtain an approximation factor < 3 might be to replace these constraints by (5.2) or (5.3) and find a different way of analyzing the approximate solution.

Problem 5.6
Is there a 2-approximation algorithm for $PD \,|\, r_{ij} \,|\, \sum w_j C_j$?

Note that the algorithm of Hall et al. [71] can cope with AND-precedence constraints on the set of jobs. If the ordering σ in Algorithm 6 respects these AND-precedence constraints, a 2-approximation for $PD \,|\, r_{ij} \,|\, \sum w_j C_j$ would also yield a 2-approximation for the single-machine problem $1 \,|\, r_j,\, prec \,|\, \sum w_j C_j$. For this problem, there was some progress in the attempt of decreasing the approximation ratio to 2 recently [155, 153], but a clean 2-approximation is still open.

Problem 5.7
Is there a 2-approximation algorithm for $1 \,|\, r_j,\, prec \,|\, \sum w_j C_j$?

Recall that $PD \,|\, r_j,\, pmtn \,|\, \sum w_j C_j$ is equivalent to the single-machine problem $1 \,|\, r_j,\, pmtn \,|\, \sum w_j C_j$ if $p_{ij} = p_j$ for all $j \in N$ and $i \in M$. The latter problem is strongly NP-hard [106], but it admits a PTAS [3]. For unit processing times, there is no need to preempt, and $1 \,|\, r_j,\, p_j = 1 \,|\, \sum w_j C_j$ is solvable in polynomial time [17]. That is, $PD \,|\, r_j,\, p_{ij} = 1 \,|\, \sum w_j C_j$ can be solved in polynomial time. However, it is not clear what the complexity of $PD \,|\, r_{ij},\, p_{ij} \in \{0,1\} \,|\, \sum w_j C_j$ is, i.e., the variant in Corollary 5.4 with individual release dates and unit processing time operations, but where a job might not have an operation on every machine.

Problem 5.8
Is there a polynomial-time algorithm for $PD \,|\, r_{ij},\, p_{ij} \in \{0,1\} \,|\, \sum w_j C_j$?

Bibliography

[1] H. M. Abdel-Wahab and T. Kameda. "Scheduling to minimize maximum cumulative cost subject to series-parallel precedence constraints". In: *Operations Research* 26.1 (1978), pp. 141–158.

[2] D Adolphson and T. C. Hu. "Optimal linear ordering". In: *SIAM Journal on Applied Mathematics* 25.3 (1973), pp. 403–423.

[3] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. "Approximation schemes for minimizing average weighted completion time with release dates". In: *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*. IEEE, 1999, pp. 32–43.

[4] R. H. Ahmadi and U. Bagchi. "Scheduling of multi-job customer orders in multi-machine environments". In: *ORSA/TIMS, Philadelphia* (1990).

[5] S. Ahmadi, S. Khuller, M. Purohit, and S. Yang. "On scheduling coflows". In: *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization*. Vol. 10328. LNCS. Springer, 2017, pp. 13–24.

[6] S. Alpern and T. Lidbetter. "Mining coal or finding terrorists: The expanding search paradigm". In: *Operations Research* 61.2 (2013), pp. 265–279.

[7] C. Ambühl and M. Mastrolilli. "Single machine precedence constrained scheduling is a vertex cover problem". In: *Algorithmica* 53.4 (2009), 488–503.

[8] C. Ambühl, M. Mastrolilli, N. Mutsanas, and O. Svensson. "On the approximability of single-machine scheduling with precedence constraints". In: *Mathematics of Operations Research* 36.4 (2011), pp. 653–669.

[9] S. Arora, L. Babai, J. Stern, and Z. Sweedyk. "The hardness of approximate optima in lattices, codes, and systems of linear equations". In: *Journal of Computer and System Sciences* 54.2 (1997), pp. 317–331.

[10] Y. Azar, I. Gamzu, and X. Yin. "Multiple intents re-ranking". In: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*. ACM, 2009, pp. 669–678.

[11] E. Balas. "On the facial structure of scheduling polyhedra". In: *Mathematical Programming Study* 24 (1985), pp. 179–218.

[12] N. Bansal. *Scheduling: Open problems old and new*. MAPSP 2017, www.mapsp2017.ma.tum.de/MAPSP2017-Bansal.pdf. last checked: June 10, 2020.

[13] N. Bansal, J. Batra, M. Farhadi, and P. Tetali. "Improved approximations for min sum vertex cover and generalized min sum set cover". In: *arXiv:2007.09172* (2020).

[14] N. Bansal, A. Gupta, and R. Krishnaswamy. "A constant factor approximation algorithm for generalized min-sum set cover". In: *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2010, pp. 1539–1545.

[15] N. Bansal and S. Khot. "Optimal long code test with one free bit". In: *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 2009, pp. 453–462.

[16] N. Bansal and S. Khot. "Inapproximability of hypergraph vertex cover and applications to scheduling problems". In: *Proceedings of the 37th International Colloquium on Automata, Languages and Programming*. Vol. 6198. LNCS. Springer, 2010, pp. 250–261.

[17] P. Baptiste. "Scheduling equal-length jobs on identical parallel machines". In: *Discrete Applied Mathematics* 103.1-3 (2000), pp. 21–32.

[18] A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, and T. Tamir. "On chromatic sums and distributed resource allocation". In: *Information and Computation* 140.2 (1998), pp. 183–202.

[19] A. Bar-Noy, M. M. Halldórsson, and G. Kortsarz. "A matched approximation bound for the sum of a greedy coloring". In: *Information Processing Letters* 71.3-4 (1999), pp. 135–140.

[20] A. Bar-Noy and G. Kortsarz. "Minimum color sum of bipartite graphs". In: *Journal of Algorithms* 28.2 (1998), pp. 339–365.

[21] R. Bar-Yehuda and S. Even. "A linear-time approximation algorithm for the weighted vertex cover problem". In: *Journal of Algorithms* 2.2 (1981), pp. 198–203.

[22] B. Braschi and D. Trystram. "A new insight into the Coffman–Graham algorithm". In: *SIAM Journal on Computing* 23.3 (1994), pp. 662–669.

[23] P. Brucker, M. R. Garey, and D. S. Johnson. "Scheduling equal-length tasks under treelike precedence constraints to minimize maximum lateness". In: *Mathematics of Operations Research* 2.3 (1977), pp. 275–284.

[24] M. Charikar, Y. Naamad, and A. Wirth. "On approximating target set selection". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Vol. 60. Leibniz International Proceedings in Informatics (LIPIcs). 2016, 4:1–4:16.

[25] C. Chekuri and R. Motwani. “Precedence constrained scheduling to minimize sum of weighted completion times on a single machine”. In: *Discrete Applied Mathematics* 98.1-2 (1999), pp. 29–38.

[26] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. “Approximation techniques for average completion time scheduling”. In: *SIAM Journal on Computing* 31.1 (2001), pp. 146–166.

[27] B. Chen, C. N. Potts, and G. J. Woeginger. “A review of machine scheduling: Complexity, algorithms and approximability”. In: *Handbook of Combinatorial Optimization*. Springer, 1998, pp. 1493–1641.

[28] B. Chen and A. P. A. Vestjens. “Scheduling on identical machines: How good is LPT in an on-line setting?” In: *Operations Research Letters* 21.4 (1997), pp. 165–169.

[29] Z.-L. Chen and N. G. Hall. *Supply chain scheduling: Assembly systems*. Tech. rep. Department of Systems Engineering, University of Pennsylvania, 2000.

[30] F. A. Chudak and D. S. Hochbaum. “A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine”. In: *Operations Research Letters* 25.5 (1999), pp. 199–204.

[31] V. Chvátal. “A greedy heuristic for the set-covering problem”. In: *Mathematics of Operations Research* 4.3 (1979), pp. 233–235.

[32] E. G. Coffman and R. L. Graham. “Optimal scheduling for two-processor systems”. In: *Acta Informatica* 1.3 (1972), pp. 200–213.

[33] S. A. Cook. “The complexity of theorem-proving procedures”. In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*. ACM, 1971, pp. 151–158.

[34] J. R. Correa and A. S. Schulz. “Single-machine scheduling with precedence constraints”. In: *Mathematics of Operations Research* 30.4 (2005), pp. 1005–1021.

[35] G. Dantzig, R. Fulkerson, and S. Johnson. “Solution of a large-scale traveling-salesman problem”. In: *Journal of the Operations Research Society of America* 2.4 (1954), pp. 393–410.

[36] R. Diestel. *Graph Theory*. Vol. 173. Graduate Texts in Mathematics. Springer, 2017.

[37] I. Dinur and S. Safra. “The importance of being biased”. In: *Proceedings on 34th Annual ACM Symposium on Theory of Computing*. ACM, 2002, pp. 33–42.

[38] I. Dinur and S. Safra. “On the hardness of approximating label-cover”. In: *Information Processing Letters* 89.5 (2004), pp. 247–254.

[39] I. Dinur and D. Steurer. "Analytical approach to parallel repetition". In: *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*. ACM, 2014, pp. 624–633.

[40] J. Du, J. Y.-T. Leung, and G. H. Young. "Scheduling chain-structured tasks to minimize makespan and mean flow time". In: *Information and Computation* 92.2 (1991), pp. 219–236.

[41] M. E. Dyer and L. A. Wolsey. "Formulating the single machine sequencing problem with release dates as a mixed integer program". In: *Discrete Applied Mathematics* 26.2-3 (1990), pp. 255–270.

[42] W. L. Eastman, S. Even, and I. M. Isaacs. "Bounds for the optimal scheduling of n jobs on m processors". In: *Management Science* 11.2 (1964), pp. 268–279.

[43] T. Erlebach, V. Kääb, and R. H. Möhring. "Scheduling AND/OR-networks on identical parallel machines". In: *Revised Papers of the 1st International Workshop on Approximation and Online Algorithms*. Vol. 2909. LNCS. Springer, 2003, pp. 123–136.

[44] U. Feige, L. Lovász, and P. Tetali. "Approximating min-sum set cover". In: *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*. Vol. 2462. LNCS. Springer, 2002, pp. 94–107.

[45] U. Feige, L. Lovász, and P. Tetali. "Approximating min sum set cover". In: *Algorithmica* 40.4 (2004), pp. 219–234.

[46] R. Fokkink, T. Lidbetter, and L. A. Végh. "On submodular search and machine scheduling". In: *Mathematics of Operations Research* 44.4 (2019), pp. 1431–1449.

[47] M. Fujii, T. Kasami, and K. Ninomiya. "Optimal sequencing of two equivalent processors". In: *SIAM Journal on Applied Mathematics* 17.4 (1969), pp. 784–789.

[48] G. W. Furnas and J. Zacks. "Multitrees: Enriching and reusing hierarchical structure". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1994, pp. 330–336.

[49] P. Gács and L. Lovász. "Khachiyan's algorithm for linear programming". In: *Mathematical Programming Study* 14 (1981), pp. 61–68.

[50] D. Gangal and A. G. Ranade. "Precedence constrained scheduling in $(2 - 7/(3p+1))\cdot$ optimal". In: *Journal of Computer and System Sciences* 74.7 (2008), pp. 1139–1146.

[51] H. L. Gantt. *Work, Wages, and Profits*. New York: Engineering Magazine Co., 1913.

[52] M. R. Garey and D. S. Johnson. "Strong NP-completeness results: Motivation, examples, and implications". In: *Journal of the ACM* 25.3 (1978), pp. 499–508.

[53] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. WH Freeman and Company, San Francisco, 1979.

[54] M. R. Garey, D. S. Johnson, and L. J. Stockmeyer. "Some simplified NP-complete problems". In: *Proceedings of the 6th Annual ACM Symposium on Theory of Computing*. ACM, 1974, pp. 47–63.

[55] N. Garg, A. Kumar, and V. Pandit. "Order scheduling models: Hardness and algorithms". In: *Proceedings of the 27th International Conference on Foundations of Software Technology and Theoretical Computer Science*. Vol. 4855. LNCS. Springer, 2007, pp. 96–107.

[56] S. Garg. "Quasi-PTAS for scheduling with precedences using LP hierarchies". In: *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming*. Vol. 107. Leibniz International Proceedings in Informatics (LIPIcs). 2018, 59:1–59:13.

[57] D. W. Gillies and J. W.-S. Liu. "Scheduling tasks with AND/OR precedence constraints". In: *SIAM Journal on Computing* 24.4 (1995), pp. 797–810.

[58] D. W. Gillies. "Algorithms to schedule tasks with AND/OR precedence constraints". PhD thesis. University of Illinois at Urbana-Champaign, 1993.

[59] M. X. Goemans. Cited as personal communication in [149]. 1996.

[60] M. X. Goemans. "A supermodular relaxation for scheduling with release dates". In: *Proceedings of the 5th International Conference on Integer Programming and Combinatorial Optimization*. Vol. 1084. LNCS. Springer, 1996, pp. 288–300.

[61] M. X. Goemans. "Improved approximation algorithms for scheduling with release dates". In: *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1997, pp. 591–598.

[62] M. X. Goemans, M. Queyranne, A. S. Schulz, M. Skutella, and Y. Wang. "Single machine scheduling with release dates". In: *SIAM Journal on Discrete Mathematics* 15.2 (2002), pp. 165–192.

[63] M. X. Goemans and D. P. Williamson. "Two-dimensional Gantt charts and a scheduling algorithm of Lawler". In: *SIAM Journal on Discrete Mathematics* 13.3 (2000), pp. 281–294.

[64] M. Goldwasser and R. Motwani. "Intractability of assembly sequencing: Unit disks in the plane". In: *Proceedings of the 5th International Workshop on Algorithms and Data Structures*. Vol. 1272. LNCS. Springer. 1997, pp. 307–320.

[65] T. F. Gonzalez and D. B. Johnson. "A new algorithm for preemptive scheduling of trees". In: *Journal of the ACM* 27.2 (1980), pp. 287–312.

[66] R. L. Graham. "Bounds for certain multiprocessing anomalies". In: *Bell System Technical Journal* 45.9 (1966), pp. 1563–1581.

[67] R. L. Graham. “Bounds on multiprocessing timing anomalies”. In: *SIAM Journal on Applied Mathematics* 17.2 (1969), pp. 416–429.

[68] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. “Optimization and approximation in deterministic sequencing and scheduling: A survey”. In: *Annals of Discrete Mathematics*. Vol. 5. Elsevier, 1979, pp. 287–326.

[69] M. Grötschel, M. Jünger, and G. Reinelt. “Facets of the linear ordering polytope”. In: *Mathematical Programming* 33.1 (1985), pp. 43–60.

[70] M. Grötschel, L. Lovász, and A. Schrijver. “The ellipsoid method and its consequences in combinatorial optimization”. In: *Combinatorica* 1.2 (1981), pp. 169–197.

[71] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. “Scheduling to minimize average completion time: Off-line and on-line approximation algorithms”. In: *Mathematics of Operations Research* 22.3 (1997), pp. 513–544.

[72] L. A. Hall and D. B. Shmoys. “Approximation schemes for constrained scheduling problems”. In: *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*. IEEE, 1989, pp. 134–139.

[73] L. A. Hall, D. B. Shmoys, and J. Wein. “Scheduling to minimize average completion time: Off-line and on-line algorithms”. In: *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1996, pp. 142–151.

[74] F. Happach. “Makespan minimization with OR-precedence constraints”. In: *arXiv:1907.08111* (2019).

[75] F. Happach, L. Hellerstein, and T. Lidbetter. “A general framework for approximating min sum ordering problems”. In: *arXiv:2004.05954* (2020).

[76] F. Happach and M. Leichter. “On the generalized min-sum set cover problem with laminar sets”. In preparation.

[77] F. Happach and A. S. Schulz. “Approximation algorithms and LP relaxations for scheduling problems related to min-sum set cover”. In: *arXiv:2001.07011* (2020).

[78] F. Happach and A. S. Schulz. “Precedence-constrained scheduling and min-sum set cover”. In: *Revised Selected Papers of the 17th International Workshop on Approximation and Online Algorithms*. Vol. 11926. LNCS. Springer, 2020, pp. 170–187.

[79] R. Hassin and A. Levin. “An approximation algorithm for the minimum latency set cover problem”. In: *Proceedings of the 13th Annual European Symposium on Algorithms*. Vol. 3669. LNCS. Springer, 2005, pp. 726–733.

[80] M. Held and R. M. Karp. “The traveling-salesman problem and minimum spanning trees”. In: *Operations Research* 18.6 (1970), pp. 1138–1162.

[81] B. Hermans, R. Leus, and J. Matuschke. “Exact and approximation algorithms for the expanding search problem”. In: *arXiv:1911.08959* (2019).

[82] D. S. Hochbaum. “Approximation algorithms for the set covering and vertex cover problems”. In: *SIAM Journal on Computing* 11.3 (1982), pp. 555–556.

[83] D. S. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Co., 1997.

[84] D. S. Hochbaum and D. B. Shmoys. “A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach”. In: *SIAM Journal on Computing* 17.3 (1988), pp. 539–551.

[85] W. A. Horn. “Single-machine job sequencing with treelike precedence ordering and linear delay penalties”. In: *SIAM Journal on Applied Mathematics* 23.2 (1972), pp. 189–202.

[86] T. C. Hu. “Parallel sequencing and assembly line problems”. In: *Operations Research* 9.6 (1961), pp. 841–848.

[87] S. Im, B. Moseley, K. Pruhs, and M. Purohit. “Matroid coflow scheduling”. In: *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming*. Vol. 132. Leibniz International Proceedings in Informatics (LIPIcs). 2019, 145:1–145:14.

[88] S. Im, M. Sviridenko, and R. van der Zwaan. “Preemptive and non-preemptive generalized min sum set cover”. In: *Mathematical Programming* 145.1-2 (2014), pp. 377–401.

[89] S. Iwata, L. Fleischer, and S. Fujishige. “A combinatorial strongly polynomial algorithm for minimizing submodular functions”. In: *Journal of the ACM* 48.4 (2001), pp. 761–777.

[90] S. Iwata, P. Tetali, and P. Tripathi. “Approximating minimum linear ordering problems”. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Vol. 7408. LNCS. Springer, 2012, pp. 206–217.

[91] J. R. Jackson. *Scheduling a production line to minimize maximum tardiness*. Tech. rep. 43. Management Science Research Project, University of California, 1955.

[92] K. Jansen. “An EPTAS for scheduling jobs on uniform processors: Using an MILP relaxation with a constant number of integral variables”. In: *SIAM Journal on Discrete Mathematics* 24.2 (2010), pp. 457–485.

[93] B. Johannes. “On the complexity of scheduling unit-time jobs with OR-precedence constraints”. In: *Operations Research Letters* 33.6 (2005), pp. 587–596.

[94] D. S. Johnson. “Approximation algorithms for combinatorial problems”. In: *Journal of Computer and System Sciences* 9.3 (1974), pp. 256–278.

[95] S. M. Johnson. “Optimal two-and three-stage production schedules with setup times included.” In: *Naval Research Logistics Quarterly* 1.1 (1954), pp. 61–68.

[96] V. Kääb. “Scheduling with AND/OR-networks”. PhD thesis. Technische Universiät Berlin, Germany, 2003.

[97] H. Kaplan, E. Kushilevitz, and Y. Mansour. “Learning with attribute costs”. In: *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*. 2005, pp. 356–365.

[98] R. M. Karp. “Reducibility among combinatorial problems”. In: *Complexity of Computer Computations*. Springer, 1972, pp. 85–103.

[99] L. G. Khachiyan. “A polynomial algorithm in linear programming (in Russian)”. In: *Doklady Akademii Nauk SSSR* 244 (1979), 1093–1096.

[100] S. Khot. “On the power of unique 2-prover 1-round games”. In: *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*. ACM, 2002, pp. 767–775.

[101] S. Khot and O. Regev. “Vertex cover might be hard to approximate to within $2-\varepsilon$”. In: *Journal of Computer and System Sciences* 74.3 (2008), pp. 335–349.

[102] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Vol. 21. Algorithms and Combinatorics. Springer, 2017.

[103] E. Kubicka and A. J. Schwenk. “An introduction to chromatic sums”. In: *Proceedings of the 17th conference on ACM Annual Computer Science Conference*. ACM. 1989, pp. 39–45.

[104] J. Kulkarni, S. Li, J. Tarnawski, and M. Ye. “Hierarchy-based algorithms for minimizing makespan under precedence and communication constraints”. In: *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2020, pp. 2770–2789.

[105] A. Kumar, R. Manokaran, M. Tulsiani, and N. K. Vishnoi. “On LP-based approximability for strict CSPs”. In: *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2011, pp. 1560–1573.

[106] J. Labetoulle, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. “Preemptive scheduling of uniform machines subject to release dates”. In: *Progress in Combinatorial Optimization*. Elsevier, 1984, pp. 245–261.

[107] S. Lam and R. Sethi. “Worst case analysis of two scheduling algorithms”. In: *SIAM Journal on Computing* 6.3 (1977), pp. 518–536.

[108] E. L. Lawler. “Sequencing jobs to minimize total weighted completion time subject to precedence constraints”. In: *Annals of Discrete Mathematics*. Vol. 2. Elsevier, 1978, pp. 75–90.

[109] E. L. Lawler. “Preemptive scheduling of precedence-constrained jobs on parallel machines”. In: *Deterministic and Stochastic Scheduling*. Springer, 1982, pp. 101–123.

[110] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons, Inc., 1985.

[111] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. “Sequencing and scheduling: Algorithms and complexity”. In: *Handbooks in Operations Research and Management Science* 4 (1993), pp. 445–522.

[112] J. K. Lenstra and A. H. G. Rinnooy Kan. “Complexity of scheduling under precedence constraints”. In: *Operations Research* 26.1 (1978), pp. 22–35.

[113] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. “Complexity of machine scheduling problems”. In: *Annals of Discrete Mathematics*. Vol. 1. Elsevier, 1977, pp. 343–362.

[114] J. K. Lenstra and D. B. Shmoys. “Elements of Scheduling”. In: *arXiv:2001.06005* (2020).

[115] J. K. Lenstra, D. B. Shmoys, and E. Tardos. “Approximation algorithms for scheduling unrelated parallel machines”. In: *Mathematical Programming* 46.1-3 (1990), pp. 259–271.

[116] J. Y.-T. Leung, H. Li, and M. Pinedo. “Order scheduling in an environment with dedicated resources in parallel”. In: *Journal of Scheduling* 8.5 (2005), pp. 355–386.

[117] J. Y.-T. Leung, H. Li, and M. Pinedo. “Scheduling orders for multiple product types to minimize total weighted completion time”. In: *Discrete Applied Mathematics* 155.8 (2007), pp. 945–970.

[118] E. Levey and T. Rothvoss. “A (1+epsilon)-approximation for makespan scheduling with precedence constraints using LP hierarchies”. In: *Proceedings of the 48th Annual ACM Symposium on Theory of Computing*. ACM, 2016, pp. 168–177.

[119] L. Lovász. “On the ratio of optimal integral and fractional covers”. In: *Discrete Mathematics* 13.4 (1975), pp. 383–390.

[120] F. Margot, M. Queyranne, and Y. Wang. “Decompositions, network flows, and a precedence constrained single-machine scheduling problem”. In: *Operations Research* 51.6 (2003), pp. 981–992.

[121] M. Mastrolilli, M. Queyranne, A. S. Schulz, O. Svensson, and N. A. Uhan. "Minimizing the sum of weighted completion times in a concurrent open shop". In: *Operations Research Letters* 38.5 (2010), pp. 390–395.

[122] J. McClintock, J. Mestre, and A. Wirth. "Precedence-constrained min sum set cover". In: *Proceedings of the 28th International Symposium on Algorithms and Computation*. Vol. 92. Leibniz International Proceedings in Informatics (LIPIcs). 2017, 55:1–55:12.

[123] R. McNaughton. "Scheduling with deadlines and loss functions". In: *Management Science* 6.1 (1959), pp. 1–12.

[124] K. Mehlhorn and P. Sanders. *Data Structures and Algorithms: The Basic Toolbox*. Springer, 2008.

[125] R. H. Möhring, M. Skutella, and F. Stork. "Scheduling with AND/OR precedence constraints". In: *SIAM Journal on Computing* 33.2 (2004), pp. 393–415.

[126] C. L. Monma. "Linear-time algorithms for scheduling on parallel processors". In: *Operations Research* 30.1 (1982), pp. 116–124.

[127] C. L. Monma and J. B. Sidney. "Sequencing with series-parallel precedence constraints". In: *Mathematics of Operations Research* 4.3 (1979), pp. 215–224.

[128] K. Munagala, S. Babu, R. Motwani, and J. Widom. "The pipelined set cover problem". In: *Proceedings of the 10th International Conference on Database Theory*. Vol. 3363. LNCS. Springer, 2005, pp. 83–98.

[129] R. R. Muntz and E. G. Coffman Jr. "Optimal preemptive scheduling on two-processor systems". In: *IEEE Transactions on Computers* 100.11 (1969), pp. 1014–1020.

[130] R. R. Muntz and E. G. Coffman Jr. "Preemptive scheduling of real-time tasks on multiprocessor systems". In: *Journal of the ACM* 17.2 (1970), pp. 324–338.

[131] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1999.

[132] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications Inc., 1998.

[133] C. H. Papadimitriou and M. Yannakakis. "On recognizing integer polyhedra". In: *Combinatorica* 10.1 (1990), pp. 107–109.

[134] C. Phillips, C. Stein, and J. Wein. "Minimizing average completion time in the presence of release dates". In: *Mathematical Programming* 82.1-2 (1998), pp. 199–223.

[135] J.-C. Picard. "Maximal closure of a graph and applications to combinatorial problems". In: *Management Science* 22.11 (1976), pp. 1268–1272.

[136] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 2016.

[137] C. N. Potts. "An algorithm for the single machine sequencing problem with precedence constraints". In: *Mathematical Programming Study* 13 (1980), pp. 78–87.

[138] C. N. Potts and V. A. Strusevich. "Fifty years of scheduling: A survey of milestones". In: *Journal of the Operational Research Society* 60.sup1 (2009), S41–S68.

[139] M. Queyranne. "Structure of a simple scheduling polyhedron". In: *Mathematical Programming* 58.1-3 (1993), pp. 263–285.

[140] M. Queyranne and A. S. Schulz. *Polyhedral approaches to machine scheduling*. Tech. rep. 408/1994. Department of Mathematics, Technical University of Berlin, 1994.

[141] M. Queyranne and A. S. Schulz. "Scheduling unit jobs with compatible release dates on parallel machines with nonstationary speeds". In: *Proceedings of the 4th International Conference on Integer Programming and Combinatorial Optimization*. Vol. 920. LNCS. Springer, 1995, pp. 307–320.

[142] M. Queyranne and Y. Wang. "Single-machine scheduling polyhedra with precedence constraints". In: *Mathematics of Operations Research* 16.1 (1991), pp. 1–20.

[143] S. Sachdeva and R. Saket. "Optimal inapproximability for scheduling problems via structural hardness for hypergraph vertex cover". In: *Proceedings of the 28th Conference on Computational Complexity*. IEEE, 2013, pp. 219–229.

[144] S. K. Sahni. "Algorithms for scheduling independent tasks". In: *Journal of the ACM* 23.1 (1976), pp. 116–127.

[145] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., 1998.

[146] A. Schrijver. "A combinatorial algorithm minimizing submodular functions in strongly polynomial time". In: *Journal of Combinatorial Theory, Series B* 80.2 (2000), pp. 346–355.

[147] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Vol. 24. Algorithms and Combinatorics. Springer, 2003.

[148] A. S. Schulz. "Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds". In: *Proceedings of the 5th International Conference on Integer Programming and Combinatorial Optimization*. Vol. 1084. LNCS. Springer, 1996, pp. 301–315.

[149] A. S. Schulz and M. Skutella. "Random-based scheduling: New approximations and LP lower bounds". In: *Proceedings of the International Workshop on Randomization and Approximation Techniques in Computer Science*. Vol. 1269. LNCS. Springer, 1997, pp. 119–133.

[150] A. S. Schulz and M. Skutella. "Scheduling-LPs bear probabilities randomized approximations for min-sum criteria". In: *Proceedings of the 5th Annual European Symposium on Algorithms*. Vol. 1284. LNCS. 1997, pp. 416–429.

[151] P. Schuurman and G. J. Woeginger. "Polynomial time approximation algorithms for machine scheduling: Ten open problems". In: *Journal of Scheduling* 2.5 (1999), pp. 203–213.

[152] J. B. Sidney. "Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs". In: *Operations Research* 23.2 (1975), pp. 283–298.

[153] R. Sitters and L. Yang. "A $(2+\varepsilon)$-approximation for precedence constrained single machine scheduling with release dates and total weighted completion time objective". In: *Operations Research Letters* 46.4 (2018), pp. 438–442.

[154] M. Skutella. "List scheduling in order of α-points on a single machine". In: *Efficient Approximation and Online Algorithms*. Vol. 3484. LNCS. Springer, 2006, pp. 250–291.

[155] M. Skutella. "A 2.542-approximation for precedence constrained single machine scheduling with release dates and total weighted completion time objective". In: *Operations Research Letters* 44.5 (2016), pp. 676–679.

[156] M. Skutella and D. P. Williamson. "A note on the generalized min-sum set cover problem". In: *Operations Research Letters* 39.6 (2011), pp. 433–436.

[157] W. E. Smith. "Various optimizers for single-stage production". In: *Naval Research Logistics Quarterly* 3.1-2 (1956), pp. 59–66.

[158] J. P. Sousa and L. A. Wolsey. "A time indexed formulation of non-preemptive single machine scheduling problems". In: *Mathematical Programming* 54.1-3 (1992), pp. 353–367.

[159] M. Streeter and D. Golovin. "An online algorithm for maximizing submodular functions". In: *Proceedings of the 22nd Annual Conference on Neural Information Processing Systems*. 2009, pp. 1577–1584.

[160] O. Svensson. "Conditional hardness of precedence constrained scheduling on identical machines". In: *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing*. ACM, 2010, pp. 745–754.

[161] W. T. Trotter. *Combinatorics and Partially Ordered Sets: Dimension Theory*. John Hopkins University Press, 1992.

[162] J. D. Ullman. “NP-complete scheduling problems”. In: *Journal of Computer and System Sciences* 10.3 (1975), pp. 384–393.

[163] G. Wang and T. C. E. Cheng. “Customer order scheduling to minimize total weighted completion time”. In: *Omega* 35.5 (2007), pp. 623–626.

[164] D. P. Williamson. “Analysis of the Held-Karp heuristic for the traveling salesman problem”. MA thesis. Massachusetts Institute of Technology, 1990.

[165] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[166] G. J. Woeginger. “On the approximability of average completion time scheduling under precedence constraints”. In: *Discrete Applied Mathematics* 131.1 (2003), pp. 237–252.

[167] L. A. Wolsey. *Mixed integer programming formulations for production planning and scheduling problems*. Invited Talk at the 12th International Symposium on Mathematical Programming. MIT, Cambridge, MA, 1985.

[168] J. Yang. “The complexity of customer order scheduling problems on parallel machines”. In: *Computers & Operations Research* 32.7 (2005), pp. 1921–1939.

[169] D. Zuckerman. “Linear degree extractors and the inapproximability of max clique and chromatic number”. In: *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*. ACM, 2006, pp. 681–690.