Parallel machine scheduling with precedence constraints

Tianyu Wang

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

CENTRALE NANTES

THESE DE DOCTORAT DE

L'ÉCOLE CENTRALE DE NANTES
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Ordonnancement*

Par
Tianyu WANG

Ordonnancement Parallèle avec Contraintes de Précédence

Thèse présentée et soutenue à Nantes, le 05/10/2018
Unité de recherche : Laboratoire des Sciences du Numérique de Nantes (LS2N)

Rapporteurs avant soutenance :

Christophe Dürr	Directeur de recherche, Sorbonne Université
Julien Moncel	Maître de conférences HDR, IUT de Roadez

Composition du Jury :

Président :	Julien Moncel	Maître de conférences HDR, IUT de Roadez
Examinateurs :	André Rossi	Professeur, Université d'Angers
	Pierre Lemaire	Maître de conférences, Grenoble INP
	Olivier Henri Roux	Professeur, Ecole Centrale de Nantes
	Christophe Dürr	Directeur de recherche, Sorbonne Université
Dir. de thèse :	Odile Bellenguez-Morineau	Maître assistant HDR, IMT Atlantique

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

CENTRALE NANTES

Titre : Ordonnancement parallèle avec contraintes de précédence

Mots clés : Ordonnancement, complexité, contraintes de précédence

Résumé : Dans cette thèse, nous considérons une famille des problèmes d'ordonnancement avec machine parallèle identique et contraintes de précédences. Ce champ de recherche fait l'objet de nombreuses études. Malgré tout, la complexité de ces problèmes varie selon de nombreux paramètres, notamment le type de graphe de précédence ou le critère retenu. De plus, il existe encore de nombreux problèmes ouverts. Nous étudions certains de ces problèmes dans cette thèse. Nous montrons notamment que le problème ouvert avec tâches de durée unitaires et graphe de précédence de type intree est NP-complet. Puis, nous prouvons que le problème avec graphe de précédence de type levelorder est NP-complet aussi. La preuve est ensuite étendue à des problèmes connexes.

Par la suite, on améliore un algorithme exponentiel pour un problème spécifique qui est NP-complet.
Enfin, nous proposons un modèle linéaire pour le problème avec contraintes de précédence quelconque, améliorant aussi les résultats de littérature.

Title : Parallel machine scheduling with precedence constraints

Keywords : Scheduling, complexity, precedence constraints

Abstract : The main problem studied in this thesis is that of parallel machine scheduling with precedence constraints. The complexity depends on the shape that the precedence graph takes and the objective function. We prove that one minimum-open problem of scheduling equal-processing-time jobs which subject to in-tree precedence constrains is NPcomplete while minimizing the total competition time. Then, we prove that the open problem of scheduling level-order precedence constrains is NP-complete too. We adapted the second proof to other scheduling problems as well.

On the other hand, we improved an exponential algorithm designed for a specific NP-hard problem. At the end, we propose a linear programming model for the general scheduling problem with arbitrary precedence constraints and processing-time. We adapt the existing models which are originally designed for other scheduling problems to parallel scheduling problem and compare these models with ours.

Acknowledgement

Foremost, I would like to express my sincere gratitude to my supervisor, Odile BELLENGUEZ-MORINEAU for her contributions of time, for her patience, for her continuous support in both academic life and paperwork, and for her trust in me. I could not have imagined having a better supervisor.

I would like to thank the jury and CST members: Christoph DÜRR, Julien MONCEL, André ROSSI, Pierre LEMAIRE, Olivier HENRI ROUX, Federico DELLA CROCE, and Christophe RAPINE, for their interest in my research and insightful comments.

I am grateful to all members of SLP with whom I have had the pleasure to work. My sincere thanks also go to all my friends in Nantes for all the fun we have had. I would also like to thank the financial support of China Scholarship Council.

I owe my deepest gratitude to my dear family for their continuous encouragement, unconditional love and support. Also, for my loving and beloved Qianyi, her faithful support is appreciated.

Contents

List of Tables

List of Figures

Part I

Introduction

The scheduling theory is widely applied in various scenarios such as manufacturing processes, practical computer systems, distribution settings, project management, etc. Thus, the scheduling problems are described as an *allocation of resources to tasks over given time periods and its goal is to optimize one or more objectives* in (Pinedo, 2012). This allocation indicates the information about the order of jobs to be executed, which is known as the sequencing problem when time is not considered. This thesis deals primarily with job-machine scheduling models, where the resources, such as real machines in workshop, processing units at a construction site, are modeled as machines and the tasks to be scheduled are modeled by jobs.

In scheduling problems, variables are usually considered as discrete. As a combinatorial optimization problem, a scheduling problem has various constraints and an objective function. In particular, constraints that we study include precedence constraints between jobs (or precedence relations in some literature). This relation between jobs can commonly be found in project management, logistics, routing, assembly flow, and networking. In precedence constraints, jobs are partially-ordered and can be represented by nodes in a directed acyclic graph (DAG). Jobs must be scheduled with respect to their priority order given by these constraints. Ordinarily, the objective function to minimize is the makespan (length) or the total completion time of all jobs. The makespan describes the utilization of machines while the total completion time reflects holding time (flow time) incurred by the schedule.

An early systematic mathematical study of scheduling problem can be traced back to the 60th (Conway et al., 1967). Then, during the past 50 years, the scheduling theory has been developed in a variety of directions:

In terms of decision time, the scheduling problem exists in two forms: static (off-line) and dynamic (on-line). In static scheduling, the data, including jobs' processing times, precedence constraints and synchronizations, are known beforehand. On the opposite in dynamic scheduling, which is also referred to as real-time scheduling, few assumptions can be made before execution, and thus, scheduling decisions must be made on-the-fly. According to the fact that we are interested in complexity, we address only the static scheduling problem in this study.

In terms of number of machines, scheduling problems have single-machine and multi-resources versions. Recently, as the development of High-performance computing (HPC) is explored, it is widely recognized that multi-machine scheduling is becoming more and more important. We consider a special case of multi-machine scheduling problem: parallel scheduling, where machines are identical, which means each one can execute all the jobs with the same speed. This assumption does not necessarily hold in real world but could be seen as a simplified version of unrelated parallel machines (RM). It is significant as a lower bound. Another extensively studied multi-machines case is shop scheduling problem, where jobs have to be processed at different stages in series. When machines are partitioned into work centers for different stages, the model is called flexible or hybrid, and is a generalization of parallel scheduling problem as well because machines in each work center are used in parallel. The related research is so abundant that some valuable results can be applied to parallel scheduling.

For the general case, when the processing time and the precedence graph are arbitrary, the parallel scheduling problem is $\mathcal{NP}$-complete, yet some specially simplified version can be optimally solved by a polynomial-time algorithm. A common simplification assumes that jobs are released at the beginning and have equal-processing-time (even unit-processing-time) and have no due date. However, the intractability also stems from the structure of the precedence graph to a large extent. A considerable amount of studies have been carried out since the 70th on the computational complexity of problems with specific precedence graphs (Ullman, 1975). They reveal that the complexity varies with the form, such as chains or trees, that the precedence graphs take. Prot and Bellenguez-Morineau (2017) surveyed these relevant results and pointed out that there exists still great number of open problems. Hence, we focus on these open problems in this thesis. This work goes on with the age-old research issue of studying scheduling problems' complexity which aims for sake of clarifying and understanding the borderline between the "hard" and "easy" cases. As all efforts to find polynomial time algorithms for the $\mathcal{NP}$-complete problems have failed, the significance is to close open problems in order to avoid wasteful attempt to optimally solve these $\mathcal{NP}$-complete problems by a polynomial algorithm, unless $P = NP$. We give the proof of $\mathcal{NP}$-completeness for some problems. The results of these works will have implications for range of practical applications which can be modeled by the parallel scheduling problem with precedence constraints.

Once a problem has been closed, a wide range of methods can be proposed to solve it. Polynomial algorithms are designed for tractable problems. Many of them choose a dispatching rule, such as the Shortest Processing Time (SPT) or Highest Level First (HLF). They are called *list scheduling* and can be used as

heuristic or even approximate methods for intractable problems. Add to this, there exist several exponential methods. For general scheduling problems, some integer-programming-based models already exist. In the second part of this thesis, we study different existing models and adaptation in order to improve resolution of parallel scheduling problem and add precedence constraints. We propose a new model which requires less variables (space) and less time for a linear-programing solver such as CPLEX. Furthermore, exponential algorithms can be used to solve optimally $\mathcal{NP}$-hard problems. These algorithms have an exponential complexity, where the exponents are widely the number of machines m. When m is fixed, the problem is polynomial solvable. We consider the problem where the precedence graph is an in-tree while minimizing the total completion time, and propose an improvement of an exact method for this problem.

In the following reminder we describe in detail the organization of this thesis:

Chapter 1 defines the notion of scheduling problems, including different precedence constraints and a sketch of complexity results in literature.

Then, in part II, we present our $\mathcal{NP}$-completeness results of different scheduling problems with unit-processing time jobs. We first introduce the 3-Partition problem. Then, two scheduling problems, $P|p_j = 1, i.t.|\sum C_j$ and $P|p_j = 1, l.o., pmtn|C_{\max}$, are separately considered in chapter 2 and 3. In the latter chapter, we adapt the proof to other scheduling problems as well.

Part III is dedicated to solve some $\mathcal{NP}$-complete scheduling problems. We first study the problem $P|p_j = p, i.t.|\sum C_j$ in chapter 4. We put forward an algorithm which is both theoretically and experimentally faster than the existing algorithm. Then, chapter 5 is developed to model the general scheduling problem $P|prec|C_{\max}$. We adapted the existing models which are originally designed for other problems, and propose a new one. We compare their performance and show that our model outperforms the others.

Finally, in part IV, we conclude by presenting a synthesis of all the results and by providing perspectives.

Chapter 1

General framework

In this thesis, we consider deterministic and off-line scheduling problems. We consider mainly parallel scheduling problem with precedence constraints. In this chapter, we first present the notations and the framework of problems we study throughout this thesis. Then, we are interested in the relation between the precedence constraints and the complexity. So, we introduce in section 1.2 the different graphs that will be considered. The general complexity result will be presented in section 1.3, while we point out the background where our research begins.

1.1 Notation

As mentioned, the scheduling problems are about allocating resources to a set of jobs over the time. We use $\mathbb{J}^T$ to represent the set of all the n jobs to be scheduled. Similarly, a blackboard bold typeface such as $\mathbb{J}$ and $\mathbb{X}$ represent always a set of jobs. We use always j to represent a job throughout the text.

The problems can be precisely described in terms of machines, constraints and objective function. Respectively, a triplet $\alpha|\beta|\gamma$, called 3-field notation, is generally used to represent scheduling problems. This notation is introduced by Graham et al. (1979) and updated by Brucker (2007).

1.1.1 Machine environment

The machine environment is described by the α-field, in the following we will consider one of those cases:

P parallel identical machines environment. Most problems studied in this thesis consider parallel machine environment, where machines have the same processing speed and no relationship between each other. The number of machines is represented by m throughout the text.

Pm The number of machines m is considered as a constant. If the variable m is chosen as a fixed parameter, some $\mathcal{NP}$-hard problem may have polynomial algorithms.

profile In some studies, such as (Dolev and Warmuth, 1985), the number of available machines varies along the time as a function. This is called **profile scheduling**.

1.1.2 Constraints environment

The constraints environment is described by β-field:

prec The precedence constraints can be represented by a directed acyclic graph (DAG): $G = (\mathbb{J}^T, E)$, where job j_1 precedes j_2 is represented by respectively $j_1 \prec j_2$ or $j_1 \rightarrow j_2$. In some literature, it is represented by a partial order $(\prec, \mathbb{J}^T)$, which is equivalent to a DAG.

Most particular structures of precedence constraints which are widely discussed are more intuitive when described by DAG, such as chains, trees, and forests to be presented in the next section, while some others are originally defined as a partial order, such as interval orders (Fishburn, 1985). As those representations are equivalent, to facilitate discussion, DAG are more generally used and also called

precedence graphs. Precisely, when the precedence graph takes a particular structure, its acronym will replace the item *prec*. We will successively consider:

1. chains (ch.)
2. in-tree (i.t.)
3. out-tree (o.t.)
4. opposing forests (o.f.)
5. level-order (l.o.)
6. series parallel (s.p.)

Those particular graphs will be detailed in section 1.2.

pmtn During a schedule, preemption may be allowed, it means a job may be preempted by other job (s) and resumed later. We assume that the preemption and resumption of jobs are immediate and take no time.

r_j Each job is given an integer release date r_j , which is the moment of time when the job arrives, and it cannot start before r_j.

$\bar{d}_j$ Each job is given a deadline $\bar{d}_j$, as its name implies, it is the maximum completion time of one job in a feasible schedule. The term **feasible** is used to describe a schedule which satisfies all the constraints.

d_j A relaxation of deadline is the due date d_j of jobs. It is not a constraint, but a *committed completion date* (Pinedo, 2012). A job that finishes after the due date is called a **tardy** job.

$p_j = p$ Each job requires a processing time, or execution time in some literature, represented by p_j. A simplified case can be considered when they have equal processing time, noted as $p_j = p$. A further simplified one is unit processing time, or UET in some literature, i.e. $p_j = 1$. Those two cases are not distinguished when the release date, due date, and deadline are integer multiples of p.

1.1.3 Objective function

The γ-field describes the objective function, which may be:

$C_{\max}$ The makespan, which is the largest completion time. The completion time of job j is note C_j, then $C_{\max} = \max_{j \in \mathbb{J}^T} C_j$. Besides, the starting time of job j in a schedule is represented as S_j.

$L_{\max}$ The maximum lateness, where the lateness of a job is defined as $L_j = C_j - d_j$ and $L_{\max} = \max_{j \in \mathbb{J}^T} L_j$.

$\sum C_j$ The total completion time, also named as total flow time or mean flow time in some literature when all jobs are released at the beginning as flow time is usually defined as $C_j - r_j$. When jobs are weighted differently, this can be written as $\sum w_j C_j$, the total weighted completion time.

$\sum U_j$ The number of tardy jobs where the binary U_j takes value 1 if and only if job j is tardy, i.e. $C_j > d_j$. When jobs are weighted differently, this can be written as $\sum w_j U_j$.

$\sum T_j$ The total tardiness, where the tardiness is defined as $T_j = \max\{L_j, 0\}$. When jobs are weighted differently, this can be written as $\sum w_j T_j$.

1.2 Precedence Graphs

In this section, we give the full definitions of the different precedence graphs that will be used in this study.

First of all, we use the term *predecessor* and *successor* only in the sense of precedence constraints, to distinguish from the jobs executed before or after in a certain schedule. We define the job which does not have a successor (predecessor) as a **final (initial)** job.

The first considered graph is **chains** (abbrv. ch), where each job has at most one direct predecessor and at most one direct successor. An **in-tree** (abbrv. i.t.) is a connected graph where each job has at most one

direct successor. In an opposite way, an **out-tree** (abbrv. o.t.) is a connected graph where each job has at most one direct predecessor.

It should be remarked that in some literature, the *in-trees* (out-trees) or *in-forests* (out-forests) is a collection of in-trees (out-trees). But by adding into this graph a dummy job which precedes (successes) all others, we combine the components[1] in a single in-tree (out-tree). It has been showed that non-delay schedule are dominant in (Weiss and Pinedo, 2012), which means solutions without enforced idle time. So, this dummy job will clearly be scheduled right after (before) the schedule of the sub-components. Then, it is clear that in-trees can be transformed into an equivalent in-tree. So, only the term in-tree (out-tree) is used in this thesis. Nevertheless, a combination of in-tree and out-tree, which is defined as an **opposing forest** (abbrv. o.f.), is worth discussing.

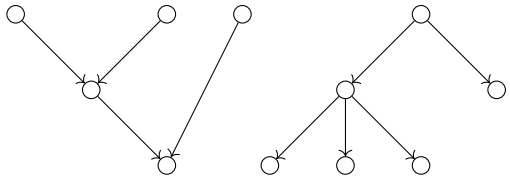

Figure 1.1: Opposing forest, where the left component is an in-tree and the right component is an out-tree

For the next definition, we consider the *level* in a graph. But it is defined using different ways in literature. It is sometimes confused with term *height*. We hereby choose the following definition:

The **height** of job j, $h(j)$ is the length of the longest chain from j to a final job. The height of the graph is the largest height of job in this graph, noted as h. The **level** of job j, $l(j)$ is recursively defined as:

$$l(j) = \begin{cases} h & \text{if } j \text{ is initial task in } G \\ \min\{l(p)|p \in pred(j)\} - 1 & \text{otherwise} \end{cases}$$

Their difference can be seen in Figure 1.2.

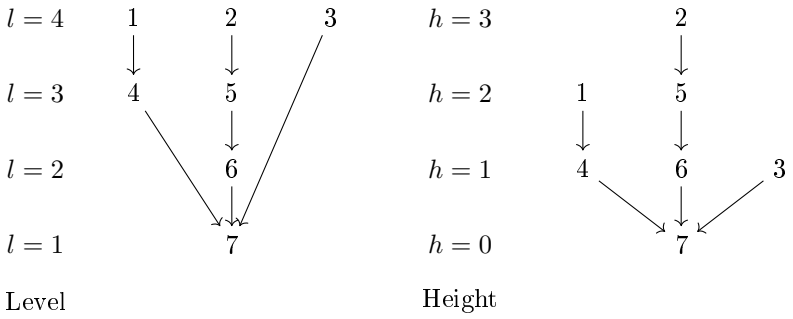

Figure 1.2: level vs height.

The term level represents the set of jobs on the same level when there is no ambiguity. Then, we may introduce the definition of a **level-order** graph, such that in each component, jobs on the same level precede all jobs of the next level.

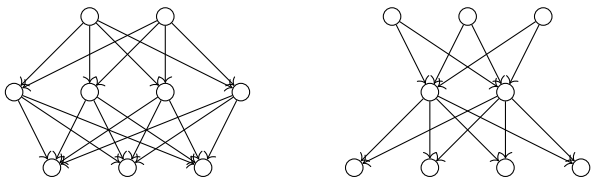

Figure 1.3: Example of a level-order graph with two components

[1] A component is a subgraph where there is a path connecting any two nodes

Finally, a more general class of graph which covers all the graphs above: **series-parallel** (abbrv. s.p.) (Lawler, 1978) .

The relation between these classes can be seen in Fig. 1.4.

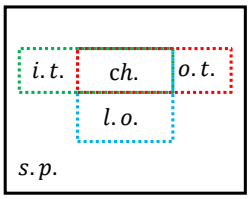

Figure 1.4: Graphs relations

1.3 Complexity result overview

In this section, we give here a general overview of complexity results and open problems. Pinedo (2012) illustrates the complexity hierarchies of scheduling problems on objective functions, see Fig. 1.5, in which $\gamma_1 \rightarrow \gamma_2$ means that minimizing γ_1 is a sub-case of minimizing γ_2. So, if it is $\mathcal{NP}$-hard to minimize γ_1, then, it is $\mathcal{NP}$-hard to minimize γ_2; if the problem can be solved for γ_2, then it can be solved for γ_1.

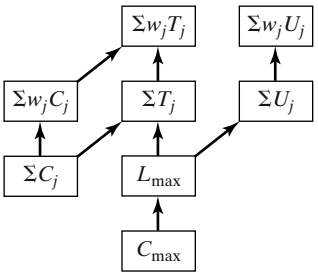

Figure 1.5: Complexity Hierarchies (Pinedo, 2012)

As it can be seen, all objective functions are generated from two basic criteria: $C_{\max}$ and $\sum C_j$. In this thesis, we study problems with these two basic objective functions for the most part.

Considering the makespan Lenstra et al. (1977) proved that the general problem, $P2||C_{\max}$, is $\mathcal{NP}$-complete even without any precedence for 2 machines while minimizing the makespan. Minimizing the total completion time seems to be more tractable: Bruno et al. (1974) proposed an optimal polynomial algorithm for a generalized problem without precedence constraints where machines are allowed to be unrelated, i.e. $R||\sum C_j$. However, when the simplest precedence constraints, chains, are added, the problem ($P2|ch|\sum C_j$ or $P2|ch|C_{\max}$) is $\mathcal{NP}$-complete no matter for total completion time or for makespan even with only two machines (Du et al., 1991).

A common simplification is to consider the unit-processing-time jobs. However, the problem $P|prec, p_j = 1|C_{\max}$ is still $\mathcal{NP}$-complete for arbitrary precedence constraints no matter whether preemption is allowed (Ullman, 1976) or not (Ullman, 1975). But when the precedence graph is chains, in-tree, out-tree, level-order or opposing forests presented in the last subsection, the complexity results may change according to objective functions.

In the following two subsections, we present the complexity results separately for minimizing the makespan and total completion time, when focusing on equal-processing-time jobs.

1.3.1 Minimizing the makespan

Now, we consider the makespan as the objective function. A well-known scheduling algorithm is proposed by Coffman and Graham (1972). It is termed as Coffman-Graham algorithm, or CG algorithm as well.

This algorithm minimizes the makespan for two machines scheduling with an arbitrary precedence graph, $P2|prec, p_j = 1|C_{\max}$. This algorithm *labels* jobs according to certain rules. Then, it executes jobs with the largest labels when there are more than m available jobs. This kind of strategy is called *list scheduling* in some literature, because it gives jobs a priority order, and then simply executes jobs according to this order. Another well-know list scheduling algorithm which labels jobs according to their height in the precedence graph is proposed by Hu (1961). It solves optimally the problem $P|i.t., p_j = 1|C_{\max}$ with unit-processing-time jobs when preemption is not allowed and the precedence graph is in-tree.

Then, as a reversed in-tree can be seen as an out-tree, Hu's algorithm can also be used to give a minimum makespan for out-tree precedence graph $P|o.t., p_j = 1|C_{\max}$. After that, we could consider the generalization $P|o.t., r_j, p_j = p|C_{\max}$ where jobs have release date and equal-processing-time. It can be optimally solved by an algorithm proposed by Brucker et al. (1977). They also proved that another generalized problem $P|o.t., p_j = p|L_{\max}$ with due dates is $\mathcal{NP}$-complete.

When it comes to the combination of in-tree and out-tree, called the opposing forest, the problem $P|o.f., p_j = 1|C_{\max}$ is proved $\mathcal{NP}$-complete by Garey et al. (1983). They proposed a polynomial algorithm when m is fixed. Another problem scheduling with level-order precedence graph, i.e. $Pm|l.o., p_j = 1|C_{\max}$ is also polynomial solvable if m is fixed (Dolev and Warmuth, 1985). However, when m is not fixed, the problem $P|l.o., p_j = 1|C_{\max}$ is still open.

Moreover, we can consider the case when preemption is allowed. The problem $P|i.t., pmtn|L_{\max}$ and $P|o.t., pmtn|L_{\max}$ are proved $\mathcal{NP}$-complete by Lawler (1982). When it comes to equal-processing-time jobs, i.e. $P|o.t., pmtn, p_j = 1|L_{\max}$, the complexity is open.

Baptiste and Timkovsky (2001) showed that preemption is advantageous[2] for minimizing the makespan. So, despite the complexity results of their non-preemptive version, problems $P|l.o., pmtn, p_j = 1|C_{\max}$, $P|o.f., pmtn, p_j = 1|C_{\max}$ and $P|o.t., pmtn, p_j = 1|L_{\max}$ are open.

1.3.2 Minimizing the total completion time

In the literature, Hu's algorithm is cited as an optimal algorithm for solving the problem with out-tree precedence graph while minimizing the total completion time, though there is no definite proof in (Hu, 1961), which is dedicated for makespan, but it can be extended. Add to this, the generalized problem $P|o.t., r_j, p_j = 1|\sum C_j$ is indeed polynomially solvable (Brucker et al., 2001).

On the other side, Hu's algorithm cannot optimally solve the problem for in-tree $P|i.t., p_j = 1|\sum C_j$ as Huo and Leung (2006) provided a counter-example. Although Baptiste et al. (2004) find an algorithm for a fixed m, it is among the *minimal open* problems according to Knust and Brucker (2009) and Prot and Bellenguez-Morineau (2017). The problem is open for level-order or opposing forests too, even when m is fixed.

Now, let us consider the problem with preemption. Preemption is useless for problem with chains precedence constraints even when jobs have arbitrary processing time (Du et al., 1991). Brucker et al. (2003) proved that it is even redundant[3] for $\sum T_j$ when there is no precedence. But for in-tree and out-tree, (Baptiste and Timkovsky, 2001) illustrated that preemption is advantageous. So, to the best of our knowledge, $P|o.f., pmtn, p_j = 1|\sum C_j$ and $P|l.o., pmtn, p_j = 1|\sum C_j$ are open.

1.3.3 Additional results

A considerable amount of researches about other graphs have been done as well.

The studies about *interval order*, where each job can be described by an interval and the precedence constraints are given by the relations between the intervals, are given by (Papadimitriou and Yannakakis, 1979), (Möhring, 1989) and (Djellab, 1999).

In *bounded height* graph, as its name suggests, the height of the graph is not larger than a given number. It is also a fixed parameter problem, where the parameter is chosen as the graph height. The studies about the complexity can be seen in (Lenstra and Rinnooy Kan, 1978) and (Dolev and Warmuth, 1984).

Some other graphs are not extensively studied. For example, *Quasi interval order* and *over interval order* are two extended version of interval order. They are defined and studied respectively in (Moukrim, 1999)

[2]Preemption is **advantageous** means that there is always preemption in an optimal schedule.

[3]Preemption is **redundant** means that there exists always an optimal schedule where there is no preemption even preemption is allowed.

β \ γ	$C_{\max}$	$\sum C_j$	$L_{\max}$
i.t.	P(Hu, 1961)	OPEN	P(Brucker et al., 1977)
l.o.	OPEN	OPEN	OPEN
o.f.	NPH(Garey et al., 1983)	OPEN	(NPH)
s.p.	(NPH)	OPEN	(NPH)
i.t.,pmtn	P(Gonzalez and Johnson, 1980)	OPEN	P(Lawler, 1982)
o.t.,pmtn	P(Lawler, 1982)	P(Brucker et al., 2001)	OPEN
l.o.,pmtn	OPEN	OPEN	OPEN
o.f.,pmtn	OPEN	OPEN	OPEN
s.p.,pmtn	OPEN	OPEN	OPEN

Table 1.1: An overview of open parallel scheduling problems with equal-processing-time jobs

and (Chardon and Moukrim, 2005). The *divide-and-conquer* graph is defined and studied in (Kubiak et al., 2009) graphs.

These precedence graphs are not studied and detailed in this thesis, but a complete survey can be seen in (Prot and Bellenguez-Morineau, 2017).

1.4 Conclusion

In this chapter, we presented a view on the state of the art of the computational complexity of the scheduling problems with specific precedence constraints and objective functions. Specific results to be used will be discussed later. According to the complexity hierarchies shown in Figure 1.5, in Table 1.1, we briefly remind the open problems with equal-processing-time jobs, i.e. $P|p_j = 1, \beta|\gamma$, where $()$ means the result derived directly from the complexity hierarchy.

One aim of the thesis is to close as much as possible open problems in this table. The last part will be dedicated to solve some $\mathcal{NP}$-hard problems.

Part II

Computational complexity

In this part, we prove the $\mathcal{NP}$-completeness of different scheduling problems with unit-processing time jobs. For the different proofs, we employ reductions from a well-known $\mathcal{NP}$-complete problem: 3-PARTITION. An instance π_{3P} of the 3-PARTITION problem is defined as hereunder (Garey and Johnson, 2002):

Definition 1. Let A be a set of $3q$ elements and B an integer, $\forall a \in A$, $\exists w_a \in Z^+$, $\frac{B}{4} < w_a < \frac{B}{2}$ and $\sum_{a \in A} w_a = qB$. The decision question is whether there is a partition $A_1, A_2, \ldots, A_q$ of A, such that $\sum_{a \in A_i} w_a = B$, $\forall i = 1, \ldots, q$.

WLOG, we assume that w_a is an integer multiple of $10q^{10}$, i.e.

$$w_a = 10q^{10} w_a^{'} \qquad w_a^{'} \in \mathbb{Q} \tag{1.1}$$

This is because when each w_a and B are multiplied by $10q^2$, the answer to the decision question does not change.

3-PARTITION is $\mathcal{NP}$-complete in the strong sense (Garey and Johnson, 1979), which enables us to construct a pseudo-polynomial transformation from π_{3P} to an instance of a given scheduling problem. This means the number of jobs n is polynomial in $B^{'}$, and therefore polynomial in B.

The reduction from 3-PARTITION to the scheduling problem consists of 2 steps. First, we transform any instance π_{3P} to a decision version of an instance of scheduling problem. Then, we prove that π_{3P} is a yes-instance if and only if the instance of scheduling problem is a yes-instance.

Chapter 2

$P|p_j = 1, i.t.|\sum C_j$

This section discuses the problem $P|p_j = 1, i.t.|\sum C_j$. We first transform any instance $\pi_{3\mathrm{P}}$ to an instance of this scheduling problem. Then, we prove the equivalence of these instances in the next two sections, which reveals the $\mathcal{NP}$-completeness of the scheduling problem.

2.1 Transformation

We propose a pseudo-polynomial transformation. So, we build an instance of the scheduling problem with in-tree precedence constraints from $\pi_{3\mathrm{P}}$. We set the number of machines $m \coloneqq 3q + 1 + B$. First, we build some dummy jobs, named x-jobs. We define

$$m_k \coloneqq B + 3k - 3 \qquad \forall k \leq q + 1$$

$$K \coloneqq B^5$$

The number of x-jobs on k_{th} level is set as $m - m_k$ and tree height is set as K^3. The corresponding tree is presented in the Figure 2.1. Those x-jobs subject to in-tree precedence constraints. In that tree, we fix for any $k \leq q + 1$, $x^1_{k-1} \prec x^1_k$, $x^2_{k-1} \prec x^1_k$ and $x^3_{k-1} \prec x^1_k$. Then, $\forall i \in \{4, 5, \ldots, m - m_{k-1}\}$, we set $x^i_{k-1} \prec x^{i-3}_k$. Thereafter, for $\forall k \in \{q + 2, q + 3, \ldots, K^3\}$, $x_{k-1} \prec x_k$, so, this part forms a chain.

Thereafter, for each $a \in A$ of $\pi_{3\mathrm{P}}$, we create two sets of jobs, which are u-jobs and v-jobs: $u^1_a, u^2_a, \ldots, u^{w_a}_a$, and $v^1_a, v^2_a, \ldots, v^{Kw_a}_a$. All the u-jobs precede directly v^1_a, i.e. $\forall i \leq w_a$, $u^i_a \prec v^1_a$. Subsequent to that, we set $v^i_a \prec v^{i+1}_a$, $\forall i < Kw_a$, so, they form a chain. The corresponding tree of the u-jobs and v-jobs for a given $a \in A$ can be observed in the Figure 2.2. As there are $3q$ elements in A, they form $3q$ in-tree components.

Thus, we transformed $\pi_{3\mathrm{P}}$ to an instance $\pi_{\text{i.t.}}$ of the considered scheduling problem. The construction is made in polynomial time of B and q (pseudo-polynomial). We build a decision version $\pi^{\sum C_j}_{\text{i.t.}}$ by asking: whether $\pi_{\text{i.t.}}$ has a schedule such that $\sum C_j \leq TCT^*$, where:

$$TCT^* \coloneqq \sum_{i=1}^{q} i(3(q + 1 - i) + 1) + \sum_{i=q+1}^{K^3} i + B \sum_{i=1}^{q} i + \sum_{a \in A} \sum_{i=1}^{Kw_a} i + KB \sum_{i=1}^{q} i$$

To establish the $\mathcal{NP}$-completeness, we hereafter prove that $\pi^{\sum C_j}_{\text{i.t.}}$ is a yes-instance if and only if $\pi_{3\mathrm{P}}$ is a yes-instance.

2.2 Proof for If

We firstly show that if $\pi_{3\mathrm{P}}$ is a yes-instance, i.e. there exists a partition $A_1, A_2, \ldots, A_q$ for $\pi_{3\mathrm{P}}$, such that $\sum_{a \in A_i} w_a = B$, then, we can build a feasible schedule (S^*) for $\pi_{\text{i.t.}}$ such that $\sum C_j \leq TCT^*$. W.l.o.g. we assume that, for each subset A_k in the partition for $k = 1, \ldots, q$, the three elements in it are a^k_1, a^k_2 and a^k_3.

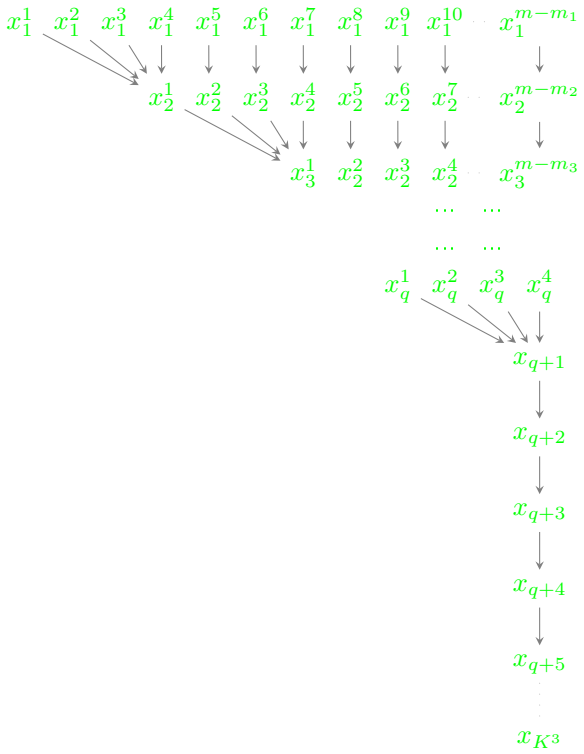

Figure 2.1: in-tree of x-jobs

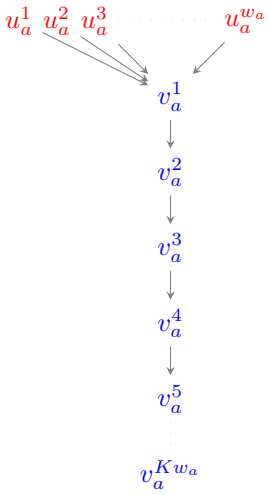

Figure 2.2: in-tree of u-jobs and v-jobs for each $a \in A$

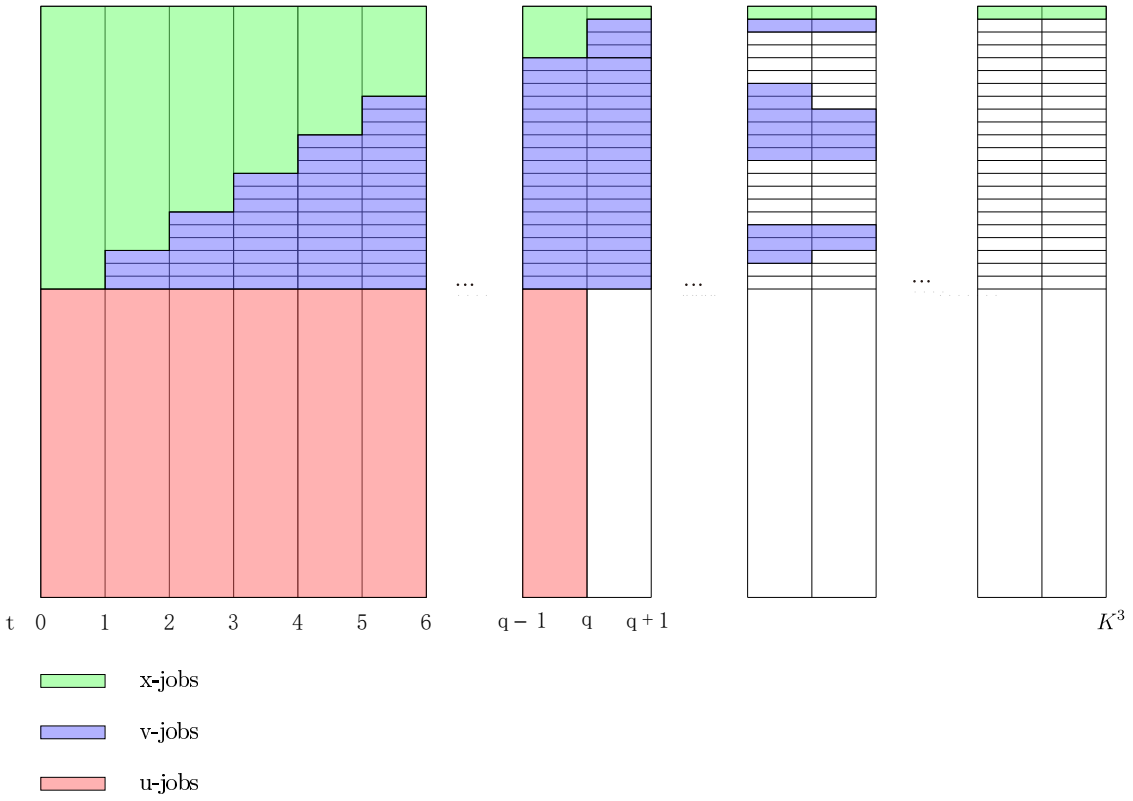

Figure 2.3: Gantt graph of (S^*)

We make use of notation TCT_x^* (resp. TCT_v^* and TCT_u^*) for representing the total completion time of x-jobs (resp. v-jobs and u-jobs) in (S^*). In the subsequent part, we show how jobs are scheduled in (S^*) together with calculating the TCT (total completion time).

Firstly, x-jobs are executed once they are **ready**, which means every x-job begins immediately after the completion time of its last-finished predecessor. Formally, $\forall i \leq q, \forall j \leq m - m_i$, $C_{x_i^j} = i$; $\forall q \leq i \leq K^3$, $C_{x_i} = i$.

Accordingly, we consider TCT for the x-jobs:

$$TCT_x^* = \sum_{i=1}^{q} \sum_{j=1}^{m-m_i} C_{x_i^j} + \sum_{i=q+1}^{K^3} C_{x_i} \tag{2.1}$$

as $\forall i \leq K^3, \forall j \leq m - m_i$, $C_{x_i^j} = i$, the first part is equivalent to:

$$\sum_{i=1}^{q} \sum_{j=1}^{m-m_i} C_{x_i^j} = \sum_{i=1}^{q} \sum_{j=1}^{m-m_i} i = \sum_{i=1}^{q} i(m - m_i)$$

and as $m - m_i = 3(q + 1 - i) + 1$, we have:

$$\sum_{i=1}^{q} \sum_{j=1}^{m-m_i} C_{x_i^j} = \sum_{i=1}^{q} i(3(q + 1 - i) + 1)$$

Then, for $i > q$, we also have $C_{x_i} = i$, so the second part of (2.1) is:

$$\sum_{i=q+1}^{K^3} C_{x_i} = \sum_{i=q+1}^{K^3} i$$

Accordingly, we can obtain the TCT of all x-jobs:

$$TCT_x^* = \sum_{i=1}^{q} i(3(q+1-i)+1) + \sum_{i=q+1}^{K^3} i \tag{2.2}$$

In the second step, considering the x-jobs, the k^{th} time-slot for $k \leq q+1$ has m_k available machines for both u-jobs and v-jobs. All the u-jobs corresponding to the subset A_k are executed in the same time-slot, the k^{th} time-slot. They need exactly B machines by definition of 3-PARTITION. Formally, $\forall k \leq q$, $\forall a \in A_k$, $\forall i \in \{1,2,3\}$, and $\forall j \leq w_{a_i^k}$, $C_{u^j_{a_i^k}} = k$. Thus, there are exactly B u-jobs executed in each time slot, i.e. $|\{u|C_u = t\}| = B, \forall t \leq q$. So, we are able to deduce that all the u-jobs are accomplished before $t = q$, and we have:

$$TCT_u^* = \sum_{t=1}^{q} t|\{u|C_u = t\}| = B\sum_{i=1}^{q} i \tag{2.3}$$

Thereafter, every v-job is executed without any delay in accordance with the precedence constraints. Formally, $\forall a \in A$, $C_{v_a^1} = C_{u_a^1} + 1$ and $\forall i \in \{2,3,\ldots,Kw_a\}$, $C_{v_a^i} = C_{v_a^{i-1}} + 1$.

So, for each v-chain corresponding to the element $a \in A$, we have $C_{v_a^i} = C_{v_a^{i-1}} + 1$, $\forall i \in \{2,3,\ldots,Kw_a\}$, that is equivalent to $C_{v_a^i} = C_{v_a^1} + i - 1$. Let u_a be any u-job preceding v_a^1, as $C_{v_a^1} = C_{u_a} + 1$, we have $C_{v_a^i} = C_{u_a} + i$. Now, we calculate the TCT for this given chain:

$$\sum_{i=1}^{Kw_a} C_{v_a^i} = \sum_{i=1}^{Kw_a} C_{u_a} + \sum_{i=1}^{Kw_a} i = Kw_a C_{u_a} + \sum_{i=1}^{Kw_a} i$$

Then, we deduce the TCT for all the v-chains, i.e. all the v-jobs:

$$TCT_v^* = \sum_{a\in A}\sum_{i=1}^{Kw_a} C_{v_a^i} = K\sum_{a\in A} w_a C_{u_a} + \sum_{a\in A}\sum_{i=1}^{Kw_a} i$$

We know the fact that, for each subset $a \in A_i$, we have $C_{u_a} = i$. So, we are able to deduce that the first part is equivalent to:

$$K\sum_{a\in A} w_a C_{u_a} = K\sum_{i=1}^{q}\sum_{a\in A_i} w_a i = K\sum_{i=1}^{q} i\sum_{a\in A_i} w_a = K\sum_{i=1}^{q} iB$$

So,

$$TCT_v^* = K\sum_{i=1}^{q} iB + \sum_{a\in A}\sum_{i=1}^{Kw_a} i \tag{2.4}$$

In k^{th} time slot of (S^*), $k < q+1$, there are exactly $m - m_k$ x-jobs, B v-jobs and $3k-3$ v-jobs. By definition of m_k, there are exactly m jobs in progress. After $t = q$, there are only one x-job and at most $3q$ v-jobs executed in parallel. Thus, there are no more than m jobs. So, this schedule respects the availability of machines.

Then, we deduce that the total completion time of (S^*), which is a feasible schedule for $\pi_{\text{i.t.}}$, is:

$$\sum C_j = TCT_v^* + TCT_u^* + TCT_x^* = TCT^*$$

2.3 Proof for Only If

Now, we prove that if $\pi_{\text{i.t.}}$ has a feasible schedule respecting the TCT^*, then π_{3P} has a partition. WLOG, let (S) be a feasible **active** schedule, defined in (Pinedo, 2012). We prove in the following part that (S) takes the same shape as (S^*) and show how to get a partition from (S).

For that purpose, we first define the TCT of u-jobs, v-jobs and x-jobs in (S) as TCT_u, TCT_v and TCT_x. Subsequent to that, we define the following gaps:

$$\Delta_u = TCT_u - TCT_u^*$$

$$\Delta_v = TCT_v - TCT_v^*$$

$$\Delta_x = TCT_x - TCT_x^*$$

In order to have (S) respecting the TCT^*, we clearly have:

$$\Delta_x + \Delta_u + \Delta_v \leq 0 \tag{2.5}$$

First of all, let us consider the x-jobs. We show that the x-jobs must have the same profile as in (S^*), which corresponds to the following lemma:

Proposition 1. *The x-jobs are scheduled once they are ready, i.e.* $\Delta_x = 0$.

Proof. By contradiction: assume that there is at least one delayed x-job. (S) is active; accordingly, this x-job can be delayed only by insertion of u-jobs or v-jobs.

Notice that the x-jobs can take no more than $m - m_1$ machines in parallel because of their precedence constraints. At any moment, there are at least $m_1 = B$ available machines for both the v-jobs and u-jobs. Moreover, in an active schedule, the makespan (C_{max}) is not larger than the number of jobs. As the total number of v-jobs and u-jobs is $KB+qB$, the last v-job or u-job finishes before or at $KB+qB$. Consequently, the delay must happen before $KB + qB$.

As the length of the final x-chain is K^3, at least $K^3 - (KB + qB)$ jobs are right-shifted. We observe that when a chain of n_c jobs is right-shifted by one time slot, then its TCT will be increased by n_c. Following this observation, we have

$$\Delta_x \geq K^3 - KB - qB$$

Remind that by definition, $TCT_u^* \ll K^2$, $TCT_v^* \ll K^2$, and $K = B^5$, $B \ll K$. Thus, we have:

$$\Delta_x \gg TCT_u^* + TCT_v^*$$

In accordance with the definition, we have $\Delta_u > -TCT_u^*$ and $\Delta_v > -TCT_v^*$, then

$$\Delta_x + \Delta_v + \Delta_u > \Delta_x - TCT_u^* - TCT_v^* \gg 0$$

This is a contradiction to (2.5). So, our assumption is false, there is no delay during the execution of the x-jobs. □

Since $\Delta_x = 0$, (2.5) becomes

$$\Delta_u + \Delta_v \leq 0 \tag{2.6}$$

Here we conclude that the number of available machines in (S) is the same as in (S^*) to schedule u-jobs and v-jobs. Now, let us deal with the u-jobs.

Definition 2. We define the u-jobs in the same component as a **u-set** and define U_t as the number of u-jobs in all the u-sets, which are completed exactly at time t.

Proposition 2. *At time* $t^* \leq q$, $\sum_{t=1}^{t^*} U_t \leq t^* B$.

Proof. The number of jobs finished at $t^* \leq q$ is at most $t^* m$ which is:

$$t^* m = t^* B + t^*(3q + 1) \leq t^* B + 3q^3$$

Considering that the number of u-jobs in a u-set is w_a, which is integer times q^5, so, the number of u-jobs in u-sets which are executed before or at t^* is at most $t^* B$. □

Now, let us move on to the v-jobs.

Definition 3. We define V_t as the number of v-jobs in the chains starting exactly at the time t for $t < q$ in (S), and V_q as the number of v-jobs in the chains starting at or after $t = q$.

Proposition 3. *In (S), at any time $t \leq q$, the total length of the v-chains beginning at t in (S) is exactly KB, i.e. $V_t = KB$.*

Proof. First, according to the precedence constraints, we have a relation between V_t and U_t:

$$\sum_{t=1}^{t^*} V_t \leq K \sum_{t=1}^{t^*} U_t \qquad \forall t^* < q$$

So, from Proposition 2, we can deduce:

$$\sum_{t=1}^{t^*} V_t \leq t^* BK \qquad \forall t^* < q$$

which is:

$$\begin{aligned} &V_1 \leq KB \\ &V_1 + V_2 \leq 2KB \\ &V_1 + V_2 + V_3 \leq 3KB \\ &\cdots \\ &\cdots \\ &V_1 + V_2 + V_3 + \cdots + V_{q-1} \leq (q-1)KB \end{aligned} \tag{2.7}$$

If we sum them up, we get:

$$(q-1)V_1 + (q-2)V_2 + \cdots + V_{q-1} \leq KB\frac{q^2 - q}{2} \tag{2.8}$$

By definition, we understand that $\sum_{t=1}^{q} V_t$ corresponds to the norm of whole set of v-jobs and it is equal to KqB:

$$\sum_{t=1}^{q} V_t = KqB \tag{2.9}$$

Then, by multiplying (2.9) by q, we get:

$$qV_1 + qV_2 + \ldots qV_q = KBq^2 \tag{2.10}$$

With (2.10)- (2.8), we get

$$V_1 + 2V_2 + \cdots + (q-1)V_{q-1} + qV_q \geq KB\frac{q^2 + q}{2}$$

which leads to conclude:

$$\sum_{i=1}^{q} iV_i \geq KB \sum_{i=1}^{q} i$$

Now, suppose by absurd that $\sum_{i=1}^{q} iV_i > KB \sum_{i=1}^{q} i$. By definition, V_i is the total length of some v-chains, which implies that V_i is integer multiple of K. So, we have:

$$\sum_{i=1}^{q} iV_i \geq KB \sum_{i=1}^{q} i + K \tag{2.11}$$

When all the v-jobs in one chain are executed without any delay, the v-jobs have the following minimum TCT:

$$TCT_v \geq \sum_{a \in A} \sum_{i=1}^{Kw_a} i + \sum_{i=1}^{q} iV_i \tag{2.12}$$

Reminding the expression of TCT_v^* in (2.4), we can deduce that:

$$\Delta_v \geq \sum_{i=1}^{q} iV_i - KB \sum_{i=1}^{q} i \tag{2.13}$$

By (2.13) and (2.11), we have $\Delta_v \geq K$. Then, by (2.6), we have $\Delta_u \leq -K$. This contradicts the fact that $\Delta_u \geq -TCT_u^* \gg -K$. Thus, we demonstrated that $\sum_{i=1}^{q} iV_i = KB \sum_{i=1}^{q} i$. So, every decomposition of group (2.7) is written with an equality, which indicates that $V_1 = V_2 = \cdots = V_q = KB$.

Finally, we can conclude that $\Delta_v = 0$ and $V_t = KB$, $\forall t \leq q$. This suggests that, at any t, the length of v-chains that start is exactly KB. □

The 3 propositions above allows us to build a partition from V_i by selecting the elements corresponding to its 3 v-chains of KB jobs. Thus, we accomplished the proof for Only If. So, we are able to reduce $\pi_{3\mathrm{P}}$ to $\pi_{\mathrm{i.t.}}^{\sum C_j}$, then, we have:

Theorem 1. *$P|p_j = 1, i.t.|\sum C_j$ is $\mathcal{NP}$-complete.*

We finished the proof for the problem with in-tree precedence constraints. The transformation and the proof in this chapter is dedicated merely to the problem $P|p_j = 1, i.t.|\sum C_j$, while in the next chapter, we move on to other scheduling problems with different precedence constraints, which requires a new reduction, where the transformation and the proof are reused.

Chapter 3

$P|p_j = 1, l.o., pmtn|C_{\max}$ and other problems

In this chapter, we begin with the problem $P|p_j = 1, l.o., pmtn|C_{\max}$. Then, we show how to adapt the proof to other open scheduling problems in the following order:

1. $P|p_j = 1, l.o.|C_{\max}$
2. $P|p_j = 1, l.o., pmtn|\sum C_j$
3. $P|p_j = 1, l.o.|\sum C_j$
4. $P|p_j = 1, o.t., pmtn|L_{\max}$
5. $P|p_j = 1, o.f., pmtn|C_{\max}$
6. $P|p_j = 1, o.f., pmtn|\sum C_j$

3.1 Transformation

We transform $\pi_{3\mathrm{P}}$ to an instance of the scheduling problem with level-order precedence graph, noted as $\pi_{\text{l.o.}}$. We set the number of machines $m = B + 2$, then build 3 sets of jobs.

First, we build a set of dummy jobs, noted x-jobs. They form a component of $qB + 1$ levels. Let the number of x-jobs on level i be $\mathcal{L}_i$, we set:

$$\mathcal{L}_i = \begin{cases} 1 & i = kB + 1, \forall k = 1, \dots, q-1 \\ 2 & i = qB + 1 \\ B + 1 & \text{otherwise} \end{cases} \tag{3.1}$$

x-jobs are subject to precedence relations where jobs on level l, which are noted $x_i^1, \dots, x_i^{\mathcal{L}_i}$, precede every job on level $l + 1$ (see Figure 3.1).

Then, the second set contains u-jobs and the third v-jobs. For each $a \in A$, we create a component of w_a u-jobs and w_a v-jobs. In each component, the u-jobs form a chain, and they precede all the v-jobs which have no precedence constraint between each other to form a unique level. Formally, we create $u_a^1 \prec u_a^2 \dots \prec u_a^{w_a} \prec v_a^k$, $\forall k = 1, \dots, w_a$, which corresponds, by definition, to a level-order graph (see Figure 3.2). As $|A| = 3q$, there are $3q$ non-dummy components.

In addition, remind that the term **profile** introduced in 1.1.1, is used to describe the number of available machines along the time.We define a specific profile, the profile $\mathcal{P}$, such as $m_t = m - \mathcal{L}_t$. This profile will be used in all this chapter. A profile of available machines is said to be filled by a set of jobs if those jobs are scheduled on the available machines without any idle time.

We have transformed $\pi_{3\mathrm{P}}$ to $\pi_{\text{l.o.}}$ by creating $(qB+1)(B+2) = O(qB^2)$ jobs, which is pseudo-polynomial. We build the first decision version $\pi_{\text{l.o.}}^{C_{\max}}$ of $\pi_{\text{l.o.}}$ by asking: whether $\pi_{\text{l.o.}}$has a schedule such that $C_{\max} \leq qB + 1$.

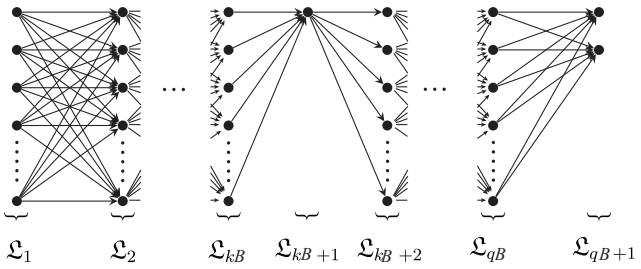

Figure 3.1: x-jobs

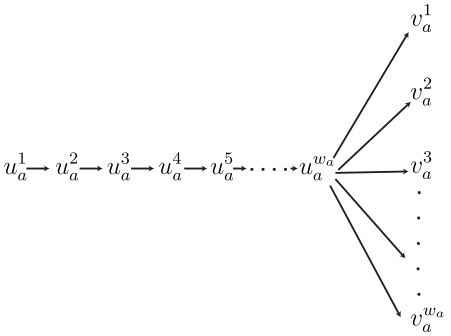

Figure 3.2: the u-jobs and v-jobs, u-jobs and v-jobs

3.2 Proof for If

In this section, we show that if π_{3P} is a yes-instance, which means it admits a partition, then $\pi_{\text{l.o.}}^{C_{\max}}$ is a yes-instance, i.e. $\pi_{\text{l.o.}}$ has a schedule such that $C_{\max} \leq qB + 1$.

We build such a schedule (S^*) by executing the x-jobs once they are ready. Formally, $C_{x_i^{i'}} = i$, $\forall i = 1, \ldots, qB + 1$ and $\forall i^{'} = 1, \ldots, \mathcal{L}_i$. The Gantt chart of (S^*) is given in Figure 3.3.

By observation, the x-jobs enforce the profile $\mathcal{P}$. Now, we show that the u-jobs and v-jobs can fill it non-preemptively:

We use $[t_1, t_2[$ to represent the half-open time interval from t_1 to t_2.$\forall i = 1, \ldots, q$,

1. In $[(i-1)B, iB - 1[$, successively execute the 3 corresponding u-chains of $A_i = \{a_1^i, a_2^i, a_3^i\}$, which corresponds to jobs: $u_{a_1^i}^1, \ldots, u_{a_1^i}^{w_{a_1^i}}, u_{a_2^i}^1, \ldots, u_{a_2^i}^{w_{a_2^i}}, u_{a_3^i}^1, \ldots, u_{a_3^i}^{w_{a_3^i}}$.

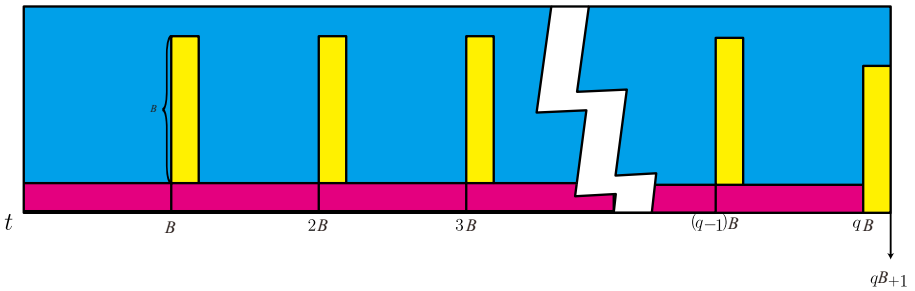

Figure 3.3: A full non-preemptive schedule (S^*) where blue (yellow, red) represents the x-jobs (resp. v-jobs, u-jobs)

2. In $[iB, iB+1[$, execute in the same time slot the corresponding v-jobs of A_i, $v^1_{a^i_1}, \ldots, v^{w_{a^i_1}}_{a^i_1}, v^1_{a^i_2}, \ldots, v^{w_{a^i_2}}_{a^i_2}, v^1_{a^i_3}, \ldots, v^{w_{a^i_3}}_{a^i_3}$.

Thus, we built a schedule which finishes all jobs exactly at $t = qB + 1$ and finish the proof.
Because the profile $\mathcal{P}$ is filled, we set the following proposition:

Proposition 4. *If π_{3P} is a yes-instance, then the u-jobs and v-jobs can fill the profile $\mathcal{P}$ non-preemptively.*

The schedule $(S)^*$ we built has no idle time. We call this a **full schedule**. In fact, a schedule such that $C_{max} = \frac{n}{m}$ is full and vice versa. So,

Proposition 5. *$\pi^{C_{\max}}_{l.o.}$ is a yes-instance if and only if $\pi_{l.o.}$ has a full schedule.*

we have also proved:

Proposition 6. *If π_{3P} is a yes-instance, then $\pi_{l.o.}$ has a full non-preemptive schedule.*

3.3 Proof for Only If

In this section, we prove that when preemption is allowed, if $\pi^{C_{\max}}_{\text{l.o.}}$ is a yes-instance, i.e. $\pi_{\text{l.o.}}$has a schedule (S) such that $C_{\max} \leq qB + 1$, then $\pi_{3\text{P}}$ is a yes-instance.

We first show that the x-jobs enforce the profile $\mathcal{P}$ in (S). By observation, any delay of the x-jobs (which means they are not executed once they are ready) or preemption resumed later will right-shift the jobs on the last level (the $qB+1^{\text{th}}$ level). It will be finished strictly after $t = qB+1$, which contradicts the makespan of (S). So, the x-jobs are executed continuously once they are ready. As the x-jobs enforce the profile $\mathcal{P}$, the u-jobs and v-jobs should fill the profile $\mathcal{P}$ to complete a full schedule.

$\forall i = 1, \ldots, q$, we define $\Phi_i = [iB, iB + \xi[$ as a half-open interval, where each machine executes at most one piece of job, ξ is a positive number as small as needed.

Define $\mathbb{V}_i$ as v-jobs executed in Φ_i, and $\mathbb{PV}_i$ as the set of u-jobs predecessors of the jobs in $\mathbb{V}_i$.

$\forall i = 1, \ldots, q$, define the **pipeline intervals**: $\mathcal{PI}_i = [(i-1)B + 1, iB[$. $\forall i = 0, \ldots, q$, define the **column intervals**: $\mathcal{CI}_i = [iB, iB + 1[$. They are noted in Figure 3.3.

We now prove that $\forall i = 1, \ldots, q$, $|\mathbb{PV}_i| = B$. We use a mathematical induction. First, we prove that this proposition holds for $i = 1$, i.e. $|\mathbb{PV}_1| = B$.

By definition, $\mathbb{PV}_1$ should be finished before $t = B$. They can only be executed in $\mathcal{CI}_0$ and $\mathcal{PI}_1$.

Lemma 1. *There are at most $3q$ u-jobs executed in parallel at any time.*

Proof. Only one u-job in the chain of each component can be executed because of the precedence of constraints and there are $3q$ components in total. □

By definition of u-jobs and Lemma 1, we have:

Lemma 2. *The amount of work of u-jobs in any column interval is at most $3q$, so, the total amount of work of u-jobs in all column intervals is at most $3q^2$.*

In addition, we can set:

Lemma 3. *The total amount of work in any pipeline interval is at most $B - 1$.*

Proof. The total amount of work in any pipeline interval, where there is only one available machine in the profile $\mathcal{P}$, cannot be greater than its length, i.e. $B - 1$. □

By Lemma 3 and 2, we know that the total amount of work of $\mathbb{PV}_1$ in $\mathcal{CI}_0$ and $\mathcal{PI}_1$ is at most $B - 1 + 3q$.

Lemma 4. *$\forall i = 1, \ldots, q$, if $|\mathbb{PV}_i| < B + 10q^2$, then $|\mathbb{PV}_i| = B$.*

Proof. We first prove that $\forall i = 1, \ldots, q$, $|\mathbb{V}_i| \geq B - 3q$ for preparation. Suppose by contradiction that $\exists i = 1, \ldots, q$, s.t. $|\mathbb{V}_i| < B - 3q$. In Φ_i, each machine executes at most one job in parallel. As the profile $\mathcal{P}$ indicates, the number of available machines for u-jobs is: $m_{iB} - |\mathbb{V}_i| > B - (B - 3q) > 3q$. So, there must be more than $3q$ u-jobs in parallel to fill the profile $\mathcal{P}$, which is a contradiction to Lemma 1. With our assumption, we can deduce that $|\mathbb{V}_i| \geq B - 3q$.

The number of u-jobs in one component, which is w_a in (1.1), is an integer multiple of $10q^2$. As $\mathbb{PV}_i$ consists only of entire u-chains, $|\mathbb{PV}_i|$ is an integer multiple of $10q^2$ too. By definition, the u-jobs and v-jobs of a component consists of the same number of u-jobs and v-jobs, using $|\mathbb{V}_i| \geq B - 3q$ we just proved, we have $|\mathbb{PV}_i| \geq |\mathbb{V}_i| \geq B - 3q > B - 10q^2$. As we have already $|\mathbb{PV}_i| < B + 10q^2$, so $B - 10q^2 < |\mathbb{PV}_i| < B + 10q^2$, the only value that $|\mathbb{PV}_i|$ can take is B. □

As we have $B + 3q - 1 \geq |\mathbb{PV}_1|$, by Lemma 4, we have $|\mathbb{PV}_1| = B$.

Now, we assume that the proposition $|\mathbb{PV}_i| = B$ holds for $i = 1, \ldots, k$, where $k = 1, \ldots, q-1$. We aim at proving that it holds for $k+1$.

As $\mathbb{PV}_1, \mathbb{PV}_2, \ldots, \mathbb{PV}_k$ are all finished no later than $t = kB$, they are executed in $\mathcal{CI}_0, \mathcal{PI}_1, \ldots, \mathcal{CI}_{k-1}, \mathcal{PI}_k$. We regroup this amount of work in two parts:

- Note the work of $\mathbb{PV}_1, \mathbb{PV}_2, \ldots, \mathbb{PV}_k$ in $\mathcal{PI}_1, \mathcal{PI}_2, \ldots, \mathcal{PI}_k$ as $W^{\leq k}_{\mathcal{PI}_{\leq k}}$;
- Note the work of $\mathbb{PV}_1, \mathbb{PV}_2, \ldots, \mathbb{PV}_k$ in $\mathcal{CI}_1, \mathcal{CI}_2, \ldots, \mathcal{CI}_{k-1}$ as $W^{\leq k}_{\mathcal{CI}_{\leq k-1}}$.

By Lemma 2, we have $W_{\mathcal{CI}} \leq 3q^2$. As the total amount is $W^{\leq k}_{\mathcal{PI}_{\leq k}} + W^{\leq k}_{\mathcal{CI}_{\leq k-1}} = |\mathbb{PV}_1| + |\mathbb{PV}_2| + \cdots + |\mathbb{PV}_k| = kB$. Thus, we have:

$$W^{\leq k}_{\mathcal{PI}_{\leq k}} \geq kB - 3q^2. \tag{3.2}$$

Then, consider $\mathbb{PV}_{k+1}$, which should be finished no later than $t = (k+1)B$. It is executed in $\mathcal{CI}_0, \mathcal{PI}_1, \mathcal{CI}_1, \ldots, \mathcal{CI}_k, \mathcal{PI}_{k+1}$. We regroup this amount of work in three parts:

- Note the work of $\mathbb{PV}_{k+1}$ in $\mathcal{PI}_1, \mathcal{PI}_2, \ldots, \mathcal{PI}_k$ as $W^{k+1}_{\mathcal{PI}_{\leq k}}$;
- Note the work of $\mathbb{PV}_{k+1}$ in $\mathcal{PI}_{k+1}$ as $W^{k+1}_{\mathcal{PI}_{k+1}}$;
- Note the work of $\mathbb{PV}_{k+1}$ in $\mathcal{CI}_0, \mathcal{CI}_1, \ldots, \mathcal{CI}_k$ as $W^{k+1}_{\mathcal{CI}_{\leq k}}$.

Similarly by Lemma 2, we have:

$$W^{k+1}_{\mathcal{CI}_{\leq k}} \leq 3q^2. \tag{3.3}$$

By Lemma 3, we have:

$$W^{k+1}_{\mathcal{PI}_{k+1}} \leq B. \tag{3.4}$$

Using Lemma 3, we can directly deduce that the total amount of works in $\mathcal{PI}_1, \mathcal{PI}_2, \ldots, \mathcal{PI}_k$ is at most kB. Remind that in (3.2), the jobs in $\mathbb{PV}_1, \mathbb{PV}_2, \ldots, \mathbb{PV}_k$ take at least $kB - 3q^2$ of these works. So, $W^{k+1}_{\mathcal{PI}_{\leq k}}$ takes at most the rest $3q^2$, which is:

$$W^{k+1}_{\mathcal{PI}_{\leq k}} \leq 3q^2. \tag{3.5}$$

See (3.3), (3.4) and (3.5) in all:

$$|\mathbb{PV}_{k+1}| = W^{k+1}_{\mathcal{CI}_{\leq k}} + W^{k+1}_{\mathcal{PI}_{k+1}} + W^{k+1}_{\mathcal{PI}_{\leq k}} \leq B + 6q^2 < B + 10q^2. \tag{3.6}$$

By Lemma 4, we have $|\mathbb{PV}_{k+1}| = B$. As we finished the mathematical induction, we proved that $\mathbb{PV}_i = B$, $\forall i = 1, \ldots, q$.

To obtain a partition of A, we only need to check the v-jobs executed in each Φ_i. The number of their predecessors is exactly B, and the elements corresponding to each Φ_i can form a partition.

Add to this, by Proposition 5, we proved that:

Proposition 7. *When preemption is allowed, if $\pi_{l.o.}$ has a full schedule, then π_{3P} is a yes-instance.*

and

Proposition 8. *When preemption is allowed, if the u-jobs and v-jobs can fill the profile $\mathcal{P}$, then π_{3P} is a yes-instance.*

As we finished the proof for both if and only if, we conclude:

Theorem 2. *$P|p_j = 1, l.o., pmtn|C_{\max}$ is $\mathcal{NP}$-complete.*

As the discussion about the preemptive case is done, we now assume that the preemption is forbidden. We aim at showing that $\pi_{3\mathrm{P}}$ and $\pi_{\mathrm{l.o.}}^{C_{\max}}$ are still equivalent. First, we set the following proposition:

Proposition 9. *$\pi_{l.o.}$ has a full non-preemptive schedule if and only if π_{3P} is a yes-instance.*

Proof. If: It can be directly given by Proposition 6.

Only If: Assume that $\pi_{\mathrm{l.o.}}$ has a full non-preemptive schedule, then of course, it can still have a (the same) full schedule when preemption is allowed. By Proposition 7, $\pi_{3\mathrm{P}}$ is a yes-instance. □

This leads directly to conclude:

Theorem 3. *$P|p_j = 1, l.o.|C_{\max}$ is $\mathcal{NP}$-complete.*

In the next section, we change the objective function to the total completion time and consider other precedence constraints graph. We do a similar transformation and adapt the proof to show the equivalence of their decision questions.

3.4 Other problems

3.4.1 Level-order graph and total completion time

In this section, we prove the considered problem is $\mathcal{NP}$-complete when the objective function is changed to the total completion time. To that purpose, we only need to build another decision version $\pi_{\mathrm{l.o.}}^{\sum C_j}$ of instance $\pi_{\mathrm{l.o.}}$ by setting the question as: whether $\pi_{\mathrm{l.o.}}$ has a schedule such that $\sum C_j \leq TCT^*$, where $TCT^* = \sum_{i=1}^{qB+1} im$.

We first discuss the preemptive case. As a preliminary step, we begin with a property of scheduling $n = km$ unit processing time jobs, where k is an integer:

Proposition 10. *Let π_* be an instance of a scheduling problem with $n = km$ unit processing time jobs where k is an integer. The decision version $\pi_*^{\sum C_j}$ asks: whether it has a schedule such that $\sum C_j \leq TCT^*$, where $TCT^* = \sum_{i=1}^{k} im$. $\pi_*^{\sum C_j}$ is a yes-instance if and only if π_* has a non-preemptive full schedule.*

Proof. If: A full non-preemptive schedule of π_* has exactly $\sum C_j = \sum_{i=1}^{k} im$.

Only If: If π_* has a non-preemptive schedule such that $\sum C_j \leq \sum_{i=1}^{k} im$, this schedule itself is a full schedule.

For the preemptive case, Brucker et al. (2003) proved that preemption is redundant for the problem $P|p_j = p|\sum T_j$, in which $P|p_j = 1|\sum C_j$ is a subcase by setting $d_j = 0$ and $p_j = 1$ for all job j. Even in $P|p_j = 1|\sum C_j$, where there is no precedence constraint, preemption cannot reduce the total completion time from a full non-preemptive schedule which is optimal for $\sum C_j$. An optimal preemptive schedule exists only when there already is a non-preemptive one. □

As $\pi_{\mathrm{l.o.}}^{\sum C_j}$ contains only unit-processing-time jobs and n is defined as $(qB+1)m$ where $qB+1$ is integer, the problem $\pi_{\mathrm{l.o.}}^{\sum C_j}$ is a yes-instance if and only if $\pi_{\mathrm{l.o.}}$ has a full non-preemptive schedule. Furthermore, Proposition 6 and 9 state that $\pi_{\mathrm{l.o.}}$ has a full non-preemptive schedule if and only if $\pi_{3\mathrm{P}}$ is a yes instance. So, we have directly $\pi_{\mathrm{l.o.}}^{\sum C_j}$ is a yes-instance if and only if $\pi_{3\mathrm{P}}$ is a yes-instance. Then, we can set:

Theorem 4. *$P|p_j = 1, l.o., pmtn|\sum C_j$ is $\mathcal{NP}$-complete.*

Then, we discuss the case when preemption is not allowed. In this case, a non-preemptive schedule such that $\sum C_j \leq \sum_{i=1}^{qB+1} im$ is equivalent to a full non-preemptive schedule too. Similarly, we have:

Theorem 5. *$P|p_j = 1, l.o.|\sum C_j$ is $\mathcal{NP}$-complete.*

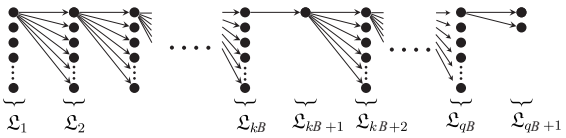

Figure 3.4: x-jobs built by out-tree

Up to the present, we have proved that for both total completion time and makespan, scheduling unit-processing-time jobs with level-order precedence constraints is $\mathcal{NP}$-complete whether it is preemptive or not. In the precedence graph we built, the u-jobs and v-jobs can be recognized not only as level-order, but also as out-trees. Based on this fact, in the following two sections, we extend this proof to scheduling problems with other precedence graphs by rebuilding only the x-jobs to respect the considered graph shape but keep the profile $\mathcal{P}$.

3.4.2 Out-trees, Maximum Lateness

Similarly, we transform $\pi_{3\mathrm{P}}$ to $\pi_{o.t.}$, an instance of the given scheduling problem with out-trees precedence constraints.

We build the u-jobs and v-jobs as previously. Their due dates are set as $qB+1$.

The x-jobs are rebuilt as following:

- $\forall i = 1, \ldots, qB+1$, build $\mathcal{L}_i$ x-jobs on the i^{th} level, where $\mathcal{L}_i$ is defined as the same as in (3.1):

$$x_1^1, \ldots x_1^{\mathcal{L}_1} \ldots\ldots x_i^1, \ldots x_i^{\mathcal{L}_i} \ldots\ldots x_{qB+1}^1, \ldots x_{qB+1}^{\mathcal{L}_{qB+1}}$$

- $\forall i^{'} = 1, \ldots, \mathcal{L}_i$, set the due date of jobs on the i^{th} level as $d_{x_i^{i'}} = i$.

- The precedence constraints are set as: $x_i^1 \prec x_{i+1}^{i^{'}}$. They can be seen as an out-tree (see Figure 3.4).

The number of machines is still $m = B+2$.

We give the decision version $\pi_{o.t.}^{L_{\max}}$ of the problem by asking whether $\pi_{o.t.}$ has a schedule whose $L_{max} \leq 0$. Then, this section is dedicated to prove its equivalence with $\pi_{3\mathrm{P}}$, i.e. it is a yes-instance if and only if $\pi_{3\mathrm{P}}$ is a yes-instance.

Proof for If

Here, we show that if $\pi_{3\mathrm{P}}$ is a yes-instance then we can create a schedule whose $L_{max} \leq 0$ for $\pi_{o.t.}$.

Execute the x-jobs once they are ready to respect the due date constraints. We observe that the x-jobs enforce the number of available machines for u-jobs and v-jobs to the profile $\mathcal{P}$. Then, according to Proposition 4 we fill the profile $\mathcal{P}$ by the u-jobs and v-jobs, in which all jobs are finished before $qB+1$, i.e. $L_{max} \leq 0$. This answers yes to $\pi_{o.t.}^{L_{\max}}$.

Proof for Only If

Let (S) be a schedule whose $L_{max} \leq 0$ for $\pi_{o.t.}$. First, consider a x-job $x_i^{i^{'}}$ for $i^{'} \leq \mathcal{L}_i$. It cannot be started before $t = i-1$ according to the precedence constraints. However, to meet its due date, it must be finished at $t = i$. These two rules force $x_i^{i^{'}}$ to get started once ready. As x-jobs enforce the profile $\mathcal{P}$, and the u-jobs and v-jobs in (S) finish before $qs+1$ to satisfy $L_{max} \leq 0$, the u-jobs and v-jobs must fill the profile $\mathcal{P}$. Then, by Proposition 8, we know that $\pi_{3\mathrm{P}}$ is a yes-instance.

As we finished the proof for both *If* and *Only If* directions, we conclude:

Theorem 6. *$P|p_j = 1, o.t., pmtn|L_{\max}$ is $\mathcal{NP}$-complete.*

Up to this point, we finish the proof for $\mathcal{NP}$-completeness of scheduling problem with out-tree precedence constraints. In the following two sections, we consider the opposing-forest while minimizing the makespan and the total completion time.

3.4.3 Opposing-forest, makespan

Now, we transform π_{3P} to $\pi_{\text{o.f.}}$, an instance of the scheduling problem with unit-processing-time jobs where the precedence constraints are opposing-forest.

We build the u-jobs and v-jobs as previously.

The x-jobs are built to compose in-trees and out-trees, see Figure 3.5.

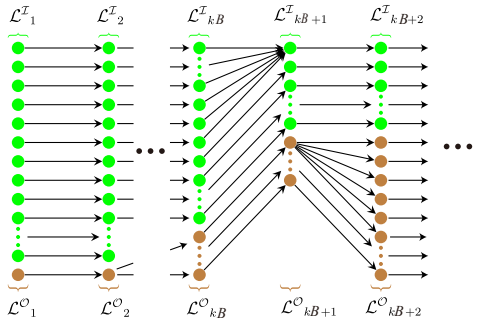

Figure 3.5: x-jobs built by an opposing-forest where green (brown) points represent an in-tree (out-tree)

First, we build an in-tree of $qB+1$ levels, such that

- for $i \leq qB+1$, recursively define

$$\mathcal{L}^{\mathcal{I}}{}_i = \begin{cases} 2qB & i = 1 \\ \mathcal{L}^{\mathcal{I}}{}_{i-1} - B & i = kB+1, \forall k = 1, 2 \ldots, q-1 \\ \mathcal{L}^{\mathcal{I}}{}_{qB} - B + 1 & i = qB+1 \\ \mathcal{L}^{\mathcal{I}}{}_{i-1} & \text{otherwise} \end{cases} \tag{3.7}$$

- On the i^{th} level, build $\mathcal{L}^{\mathcal{I}}{}_i$ jobs $x^1_{\mathcal{I},i}, \ldots x^{\mathcal{L}^{\mathcal{I}}{}_i}_{\mathcal{I},i}$
- set the precedence constraints as:
  - if $\mathcal{L}^{\mathcal{I}}{}_i = \mathcal{L}^{\mathcal{I}}{}_{i-1}$, $x^{i'}_{\mathcal{I},i-1} \prec x^{i'}_{\mathcal{I},i}$, for $i' = 1, \ldots, \mathcal{L}^{\mathcal{I}}{}_i$
  - if $\mathcal{L}^{\mathcal{I}}{}_i < \mathcal{L}^{\mathcal{I}}{}_{i-1}$, let $\Delta = \mathcal{L}^{\mathcal{I}}{}_{i-1} - \mathcal{L}^{\mathcal{I}}{}_{i-1} + 1$, then set $x^{i'}_{\mathcal{I},i-1} \prec x^1_{\mathcal{I},i}$, for $i' = 1, \ldots, \Delta$; and $x^{i'+\Delta}_{\mathcal{I},i-1} \prec x^{i'}_{\mathcal{I},i}$ for $i' = 1, \ldots, \mathcal{L}^{\mathcal{I}}{}_i - \Delta$.

Then, we build an out-tree of $qB+1$ levels

- for $i \leq qB+1$, recursively define

$$\mathcal{L}^{\mathcal{O}}{}_i = \begin{cases} 1 & i = 1 \\ \mathcal{L}^{\mathcal{O}}{}_{i-1} + B & i = kB+2, \forall k = 1, 2 \ldots, q-1 \\ \mathcal{L}^{\mathcal{O}}{}_{i-1} & \text{otherwise} \end{cases} \tag{3.8}$$

- On the i^{th} level, build $\mathcal{L}^{\mathcal{O}}{}_i$ jobs $x^1_{\mathcal{O},i}, \ldots x^{\mathcal{L}^{\mathcal{O}}{}_i}_{\mathcal{O},i}$
- set the precedence constraints as:
  - if $\mathcal{L}^{\mathcal{O}}{}_i = \mathcal{L}^{\mathcal{O}}{}_{i-1}$, $x^{i'}_{\mathcal{O},i-1} \prec x^{i'}_{\mathcal{O},i}$, $i' = 1, \ldots, \mathcal{L}^{\mathcal{I}}{}_i$
  - if $\mathcal{L}^{\mathcal{O}}{}_i < \mathcal{L}^{\mathcal{O}}{}_{i-1}$, let $\Delta = \mathcal{L}^{\mathcal{O}}{}_i - \mathcal{L}^{\mathcal{O}}{}_{i-1} + 1 = B+1$, then set $x^1_{\mathcal{O},i-1} \prec x^{i'}_{\mathcal{O},i}$, for $i' \leq \Delta$; and $x^{i'}_{\mathcal{O},i-1} \prec x^{i'+\Delta}_{\mathcal{O},i}$ for $i' = 1, \ldots \mathcal{L}^{\mathcal{O}}{}_{i-1} - \Delta$.

In this instance, the number of machines is set to $m = 2qB + 2$. We compare number of x-jobs on the i^{th} level ${\mathcal{L}^{\mathcal{O}}}_i + {\mathcal{L}^{\mathcal{I}}}_i = \mathcal{L}_i + \delta$, where $\delta = 2qB + 2 - (B + 2) = 2qB - B$. It is exactly the difference between m defined in $\pi_{\text{o.f.}}$ and $\pi_{\text{l.o.}}$. We find that scheduling the x-jobs once they are ready still enforces the profile $\mathcal{P}$ of available machines for u-jobs and v-jobs and the previous propositions about the v-jobs, u-jobs and the profile $\mathcal{P}$ still yield.

The first decision version $\pi_{\text{o.f.}}^{C_{\max}}$ asks: whether $\pi_{\text{o.f.}}$ has a feasible schedule such that $C_{\max} \leq qB + 1$. Then, we prove its equivalence with $\pi_{3\text{P}}$ by showing that $\pi_{3\text{P}}$ is a yes-instance if and only if $\pi_{\text{o.f.}}^{C_{\max}}$ is a yes-instance.

Proof for If

As $\pi_{\text{o.f.}}$ contains $(qB+1)m$ jobs, then $\pi_{\text{o.f.}}^{C_{\max}}$ is a yes-instance if and only if $\pi_{\text{o.f.}}$ has a full schedule. In this subsection, we set that:

Proposition 11. *If π_{3P} is a yes-instance, then $\pi_{o.f.}$ has a full non-preemptive schedule.*

Proof. Using Proposition 4, as $\pi_{3\text{P}}$ is a yes-instance, the u-jobs and v-jobs can fill the profile $\mathcal{P}$ non-preemptively. Thus, we can build a schedule by letting the x-jobs be executed once they are ready and filling the profile $\mathcal{P}$ with the u-jobs and v-jobs.

As all jobs are finished before $qB+1$, and there are exactly $(qB+1)m$ jobs in $\pi_{\text{o.f.}}$, this is a full schedule without preemption.

□

Proof for Only If

In this subsection, we prove that if $\pi_{\text{o.f.}}$ has a full schedule, then $\pi_{3\text{P}}$ is a yes-instance. We begin with studying the u-jobs and v-jobs and the profile $\mathcal{P}$.

First, by observation, whether in the in-tree or in the out-tree, every x-job has a successor in the $(qB+1)^{\text{th}}$ level. So, any delay will right-shift this job. It will be finished strictly after $t = qB + 1$, which contradicts the makespan $qB + 1$. So, the x-jobs are executed once ready and enforce the profile $\mathcal{P}$. Thus, the u-jobs and v-jobs must fill the profile $\mathcal{P}$ to finish before $qB + 1$.

Then, by Proposition 8, we have:

Proposition 12. *When preemption is allowed, if $\pi_{o.f.}$ has a full schedule, then π_{3P} is a yes-instance.*

Proposition 11 and 12 bring out:

Theorem 7. *$P|p_j = 1, o.f., pmtn|C_{\max}$ is $\mathcal{NP}$-complete.*

If $\pi_{\text{o.f.}}$ has a non-preemptive full schedule, then it has one full schedule when preemption is allowed, so by Proposition 12, we have:

Proposition 13. *If $\pi_{o.f.}$ has a non-preemptive full schedule, then π_{3P} is a yes-instance.*

We move to another objective function, the total completion time, and prove that the problem is still $\mathcal{NP}$-complete.

3.4.4 Opposing-forest, total completion time

Considering the total completion time, we build another decision version $\pi_{\text{o.f.}}^{\sum C_j}$ of $\pi_{\text{o.f.}}$. We set the decision question as: whether $\pi_{\text{o.f.}}$ has a feasible schedule such that $\sum C_j \leq TCT^*$, where $TCT^* = \sum_{i=1}^{qB+1} im$.

As $\pi_{\text{o.f.}}$ contains $(qB+1)m$ jobs, when preemption is allowed, by Proposition 10, $\pi_{\text{o.f.}}^{\sum C_j}$ is a yes-instance if and only if $\pi_{\text{o.f.}}$ has a full non-preemptive schedule. Furthermore, Proposition 11 and 13 show that $\pi_{\text{o.f.}}$ has a full non-preemptive schedule if and only if 3-PARTITION is a yes-instance, which brings out:

Theorem 8. *$P|p_j = 1, o.f., pmtn|\sum C_j$ is $\mathcal{NP}$-complete.*

β \ γ	$C_{\max}$	$\sum C_j$	$L_{\max}$
i.t.	P(Hu, 1961)	NPH	P(Brucker et al., 1977)
l.o.	NPH	NPH	(NPH)
o.f.	NPH(Garey et al., 1983)	(NPH)	(NPH)
s.p.	(NPH)	(NPH)	(NPH)
i.t.,pmtn	P(Gonzalez and Johnson, 1980)	OPEN	P(Lawler, 1982)
o.t.,pmtn	P(Lawler, 1982)	P(Brucker et al., 2001)	NPH
l.o.,pmtn	NPH	NPH	(NPH)
o.f.,pmtn	NPH	NPH	NPH
s.p.,pmtn	(NPH)	(NPH)	(NPH)

Table 3.1: An updated overview of complexity of $P|p_j = 1, \beta|\gamma$

3.5 Conclusion

In this chapter, we first put forward a proof of $\mathcal{NP}$-completeness for $P|p_j = 1, \text{in-tree}|\sum C_j$. Then, we proved that scheduling unit-processing-time jobs under level-order or opposing-forests precedence constraints is $\mathcal{NP}$-complete, for both $C_{\max}$ and $\sum C_j$ whether or not preemptive. Scheduling out-trees is shown to be $\mathcal{NP}$-complete by using similar proof when minimizing $L_{\max}$. This work updates the earlier study of scheduling problems' complexity in literature. The Table 1.1 can be updated. New complexity results of parallel scheduling unit-processing-time jobs under different precedence constraints are shown in Table 3.1.

Another result comes from the complexity hierarchy. When the criterion is the total tardiness $\sum T_i = \sum \max\{L_{\max}, 0\}$ or the total number of tardy jobs $\sum U_j$ (binary U_j equals 1 if and only if j is tardy), our result also implies that $P|o.t., p_j = 1, pmtn|\sum T_j$ and $P|o.t., p_j = 1, pmtn|\sum U_j$ are $\mathcal{NP}$-complete as well. With this hierarchy, the results can be extended to other open problem not included in Table 3.1.

The table also shows that when preemption is allowed, the complexity of the problem $P|p_j = 1, i.t., pmtn|\sum C_j$ is still open. However, the preemption is not redundant to reduce the total completion timeand previous results cannot be directly extended.

Despite the fact that preemption may reduce the total completion time, we still conjecture that, with the same transformation, the equivalence of $\pi_{3\text{P}}$ and $\pi_{\text{i.t.}}^{\sum C_j}$ still holds even if preemption is allowed, and the problem shall be $\mathcal{NP}$-complete as well.

Considering the fixed parameter m, there are still some other problems whose complexity is open, such as $Pm|p_j = 1, l.o., pmtn|C_{\max}$. Our next research avenue may be dedicated to study these open problems.

As the problems are $\mathcal{NP}$-complete, they have no polynomial algorithm unless $P = NP$. In the next part, we focus on how to exponentially solve these problems.

Part III

Problem solving

In this part, we focus on how the scheduling problems we study can be solved. We put forward an algorithm for the specific problem $P|p_j = p, i.t.|\sum C_j$, which is both theoretically and experimentally faster than the existing algorithm for this problem. Then, we study the general problem $P|prec|C_{\max}$ by integer programming models. We adapted the existing models which are originally designed for other problems, and propose a new one. We compare their performance and show that our model outperforms the others.

Chapter 4

An improved algorithm for $P|p_j = p, i.t.|\sum C_j$

In this chapter, we study how to solve the problem $P|p_j = p, i.t.|\sum C_j$, which deals with in-tree precedence graph while minimizing the total completion time. We proved in Chapter 2 that the problem is $\mathcal{NP}$-hard. However, when the number of machines m is fixed, the problem, written as $Pm|p_j = p, i.t.|\sum C_j$, can be polynomially solved in $O(n^m)$ by an algorithm by Baptiste et al. (2004). We here propose a new algorithm in $O(h^m)$ where h is the height of the tree, and show that it is also experimentally faster than Baptiste's algorithm.

First, we introduce the state of the art in the following section. Second, we propose and prove a theorem, on which the optimality of our algorithm is based. Third, we illustrate the algorithm with an example. Then, we compare it to Baptiste's algorithm and present the experiment results.

4.1 Previous studies

This section is dedicated to introduce the work of Hu (1961) and Baptiste et al. (2004), that will be useful for next parts.

Let us recall that in (Hu, 1961) the *height* of job j is defined as the length of longest chain from j to the final job, noted as $h(j)$. He proved that an algorithm who executes the *highest* jobs (with largest $h(j)$) when there are more than m candidates, is optimal for makespan, i.e. $P|p_j = p, i.t.|C_{\max}$.

On the other hand, Baptiste et al. (2004) showed that an optimal schedule who minimizes the total flow time always takes the shape described in Fig. 4.1. The set of jobs in the first incomplete slot is noted as $\mathbb{J}^*$. $\mathbb{J}^*$ separates jobs into 2 parts. As the second part $\mathbb{J}^2$ is composed of successors of $\mathbb{J}^*$, when $\mathbb{J}^*$ is given, the 2 parts can be immediately obtained. Thus, an optimal schedule can be *generated* in polynomial time from $\mathbb{J}^*$: the full partial schedule of the first part can be obtained by Hu's algorithm, and any active partial schedule of $\mathbb{J}^2$ is optimal. To obtain $\mathbb{J}^*$, they proposed to check all sets of at most $m-1$ jobs, which requires $O(n^m)$ iterations. When m is fixed, this method finds an optimal schedule in polynomial time.

4.2 Further analysis

We aim at a more efficient way to find a $\mathbb{J}^*$ instead of checking all combinations. In this section, we prove that there exists always a $\mathbb{J}^*$ who satisfies some properties.

4.2.1 Definitions

We first formally define different sets of jobs in an optimal schedule. As $\mathbb{J}^*$ is the set of jobs in the first slot with less than m jobs, and $\mathbb{J}^2$ is its successors, we define $\mathbb{J}^{'}$ as the set of jobs in the previous slot of $\mathbb{J}^*$. Then, let $\mathbb{J}^1$ be the jobs executed before $\mathbb{J}^{'}$.

We partition set $\mathbb{J}^*$ into $\mathbb{A}^*$ and $\mathbb{B}^*$: each job in $\mathbb{B}^*$ has predecessor in $\mathbb{J}^{'}$, and jobs in $\mathbb{A}^*$ do not. Reversely, we define $\mathbb{B}^{'} \subset \mathbb{J}^{'}$, the set of predecessors of $\mathbb{B}^*$, and $\mathbb{A}^{'} := \mathbb{J}^{'} \setminus \mathbb{B}^{'}$. See Fig. 4.1.

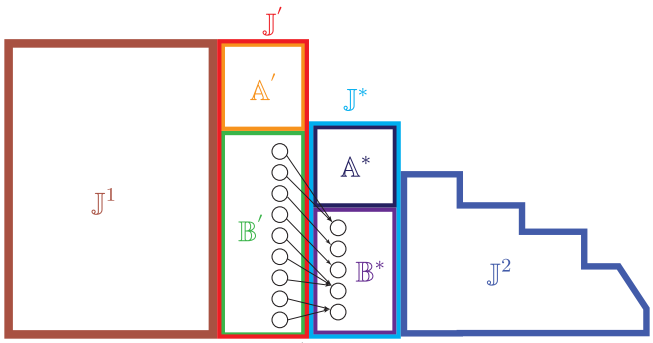

Figure 4.1: Profile of an optimal schedule

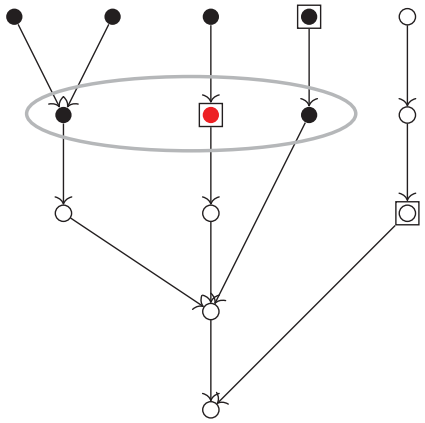

Figure 4.2: The in-tree graph. Job j is in red, set $\mathbb{X}$ is in box $\square$, $\mathbb{L}_{\mathbb{X}}(j) = \emptyset$, $\mathbb{P}_{\mathbb{X}}(j)$ is solid, and $\mathbb{S}_{\mathbb{X}}(j)$ is in the oval.

Then, we recall the *level* of a job introduced in section 1.2: h minus the length of the longest chain from an initial job to j, represented by $l(j)$ to distinguish from $h(j)$, the length of the longest chain from j to a final job. Their difference can be seen in Fig. 1.2. The terms *high* (higher, highest) and *low* (lower, lowest) are used to describe $l(j)$ from now on.

The formal definition of $l(j)$ is:

$$l(j) \coloneqq \begin{cases} h & \text{if } j \text{ is initial task in } G \\ \min\{l(p)|p \in pred(j)\} - 1 & \text{otherwise} \end{cases}$$

Then, we consider 3 different operators on a job j and a set $\mathbb{X}$. They return a set of jobs.

1. The set of jobs in $\mathbb{X}$ which are strictly lower than j :

$$\mathbb{L}_{\mathbb{X}}(j) \coloneqq \{j^* \in \mathbb{X}|l(j^*) < l(j)\}$$

2. A pruned tree:

$$\mathbb{P}_{\mathbb{X}}(j) \coloneqq \{j_1 \in \mathbb{J}^T|l(j_1) > l(j_2), \forall j_2 \in \mathbb{L}_{\mathbb{X}}(j)\} \setminus pred(\mathbb{L}_{\mathbb{X}}(j))$$

It removes all predecessors of $\mathbb{L}_{\mathbb{X}}(j)$ from jobs strictly higher than jobs in $\mathbb{L}_{\mathbb{X}}(j)$.

3. The set of jobs in $\mathbb{P}_{\mathbb{X}}(j)$ which are on the same level as j :

$$\mathbb{S}_{\mathbb{X}}(j) \coloneqq \{j^s|l(j^s) = l(j), j^s \in \mathbb{P}_{\mathbb{X}}(j)\}$$

See Fig. 4.2 for an illustration.

$\forall j \in \mathbb{X}$, if $|\mathbb{S}_{\mathbb{X}}(j)| \leq k$, then j is called **k-limited** in $\mathbb{X}$, where k is a positive number. Furthermore, a set $\mathbb{X}$ is **pure-k-limited** when all jobs in it are k-limited.

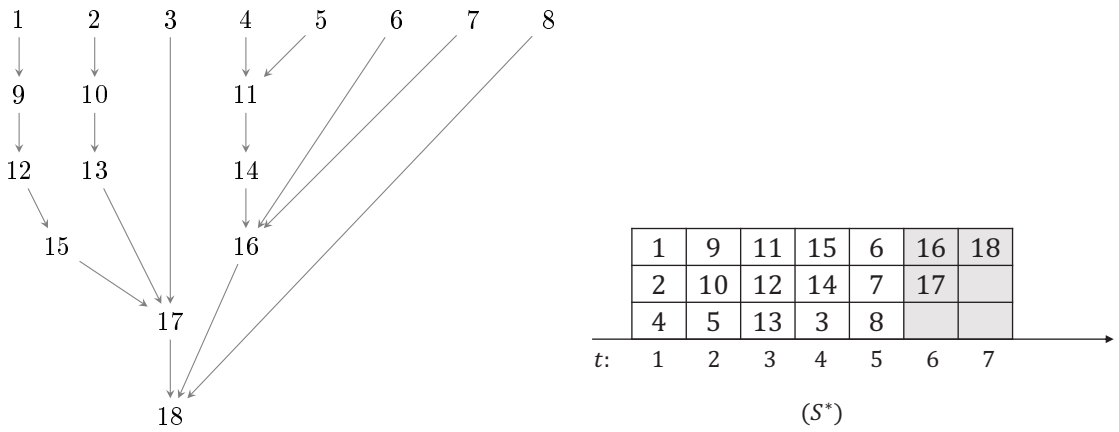

Figure 4.3: example: $\mathbb{J}^T$ in graph G and an optimal schedule (S^*)

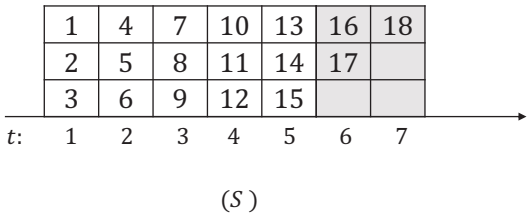

Figure 4.4: example: (S)

Finally, a set $\mathbb{X} \subset \mathbb{J}^T$ is defined as **regular** if either it has a pure-m-limited superset $\mathbb{V}$, such that $|\mathbb{V}| \leq m$; or the jobs in $\mathbb{X}$ can be partitioned into 3 groups:

Group 1 in which jobs are m-limited in $\mathbb{X}$, i.e. $|\mathbb{P}_{\mathbb{X}}(j)| \leq m$.

Group 2 in which jobs are $2m$-limited in $\mathbb{X}$ and admit the same level, noted as $l^{\text{Group 2}}$, i.e. $\forall j \in$Group 2, $|\mathbb{P}_{\mathbb{X}}(j)| \leq 2m$ and $l(j) = l^{\text{Group 2}}$.

Group 3 in which each job j is the lexicographically first one in $\mathbb{S}_{\mathbb{X}}(j)$, and $l(j) \geq l^{\text{Group 2}}$.

Using this definition, we set:

Theorem 9. *For any instance of $P|p_j = p, i.t.|\sum C_j$, there exist an optimal schedule with a regular $\mathbb{J}^*$.*

For convenience, a schedule is also called a regular schedule if $\mathbb{J}^*$ is regular. The following of this section is dedicated to prove the Theorem, i.e. the existence of a regular $\mathbb{J}^*$. To show the existence of a regular $\mathbb{J}^*$, we first build a particular schedule (S) from an arbitrary optimal one (S^*). Then, we prove that the $\mathbb{J}^*$ of (S) is regular. W.l.o.g. we assume that $\mathbb{J}^1 \cup \mathbb{J}^{'} \neq \emptyset$, otherwise any active schedule is optimal.

4.2.2 Construction of regular optimal schedule (S) from an arbitrary optimal schedule (S^*)

Let (S^*) be any optimal solution for $P|p_j = p, i.t.|\sum C_j$.

Step 1: Build the reverse graph $G^R = \{\mathbb{J}^{'} \cup \mathbb{J}^1, E^R\}$. Set $j_1 \prec^R j_2, \forall j_2 \prec j_1$, where $\prec^R$ means edges in E^R. An example can be seen in Fig. 4.3 and the reverse graph of $\mathbb{J}^{'} \cup \mathbb{J}^1$ in Fig. 4.5.

Step 2: Use Hu's algorithm to get a full schedule (S_1) for jobs in G^R. For jobs of the same height in G^R, the priority is given to those who have a successor in J^* in G. Jobs with same priority are executed in lexicographical order.

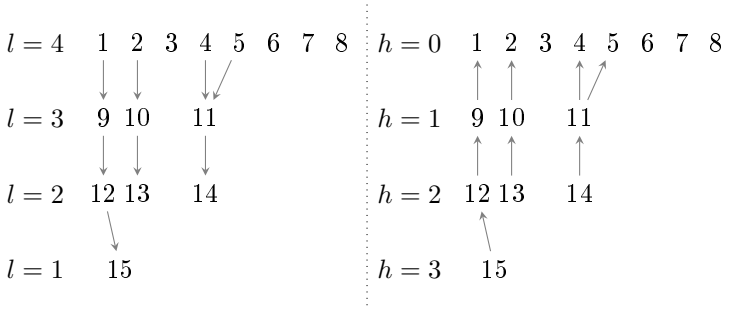

Figure 4.5: example: $\mathbb{J}^{'} \cup \mathbb{J}^1$ in G and its reverse graph G^R. The relation of height and level.

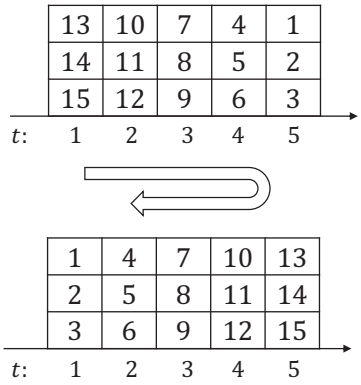

Figure 4.6: example: schedule by Hu's algorithm and its reverse schedule

Step 3: Reverse (S_1). See Fig. 4.6.

Step 4: Replace the sub-schedule of $\mathbb{J}^{'} \cup \mathbb{J}^1$ in (S^*) by (S_1). See Fig. 4.4.

Step 5: $\forall j_{\mathbb{A}^*} \in \mathbb{A}^*$, $\forall j_{\mathbb{J}'\mathbb{J}^1} \in \mathbb{J}^{'} \cup \mathbb{J}^1$, if

1. $l(j_{\mathbb{J}'\mathbb{J}^1}) < l(j_{\mathbb{A}^*})$, or $l(j_{\mathbb{J}'\mathbb{J}^1}) = l(j_{\mathbb{A}^*})$ but $j_{\mathbb{J}'\mathbb{J}^1}$ lexicographically precedes $j_{\mathbb{A}^*}$, and
2. $pred(j_{\mathbb{A}^*}) = \emptyset$, or $pred(j_{\mathbb{A}^*})$ finishes before $S_{j_{\mathbb{J}'\mathbb{J}^1}}$, and
3. $succ(j_{\mathbb{J}'\mathbb{J}^1}) = \emptyset$, or $succ(j_{\mathbb{J}'\mathbb{J}^1})$ begins after $C_{j_{\mathbb{A}^*}}$,

then, switch $j_{\mathbb{J}'\mathbb{J}^1}$ and $j_{\mathbb{A}^*}$. Go back to Step 1.

After these operations, the schedule (S^*) is renamed as (S). Notice that Hu's algorithm is optimal for makespan and out-tree. The reverse schedule of a feasible schedule of the reverse graph is feasible. The switch in Step 5 respects the precedence constraints. The profile of the Gantt chart is unchanged, (S) is still optimal. Finally, we underline that the go-back-loop finishes within n^2 repetitions because every time it moves the level of a jobs in $\mathbb{A}^*$ down.

4.2.3 Properties of (S)

Before completing proof of the Theorem, we introduce the following properties, which will be used to prove the regularity of (S).

Propriety 1. *If $\mathbb{A}^* \neq \emptyset$ and $\mathbb{A}^{'} \neq \emptyset$, then $\forall j_{\mathbb{A}^*} \in \mathbb{A}^*$, $\forall j_{\mathbb{A}^{'}} \in \mathbb{A}^{'}$, $l(j_{\mathbb{A}^{'}}) \geq l(j_{\mathbb{A}^*})$.*

Proof. By definition, jobs in $\mathbb{A}^*$ have no predecessor in $\mathbb{J}^{'}$ and jobs in $\mathbb{A}^{'}$ have no successor in $\mathbb{J}^*$. If $l(j_{\mathbb{A}^{'}}) < l(j_{\mathbb{A}^*})$, $j_{\mathbb{A}^{'}}$ and $j_{\mathbb{A}^*}$ should have been switched in Step 5. □

Propriety 2. *$\forall j_{\mathbb{J}'} \in \mathbb{J}^{'}$, $\forall j_{\mathbb{J}'\mathbb{J}^1} \in \mathbb{J}^1 \cup \mathbb{J}^{'}$ if $l(j_{\mathbb{J}'}) > l(j_{\mathbb{J}'\mathbb{J}^1})$ and $j_{\mathbb{J}'\mathbb{J}^1}$ does not have a successor in $\mathbb{J}^{'}$, then $j_{\mathbb{J}'\mathbb{J}^1} \in \mathbb{J}^{'}$.*

Proof. Separately consider two cases:

Case 1: $j_{\mathbb{J}'\mathbb{J}^1}$ does not have a successor in $\mathbb{J}^1$. Let $j^{'}$ be the image of job j in the reverse graph G^R. We find a relationship between l and h:

$$l(j) + h(j^{'}) = h \tag{4.1}$$

This relationship is clearly shown in Fig. 4.5.

As $j_{\mathbb{J}'\mathbb{J}^1}$ does not have a successor in $\mathbb{J}^1 \cup \mathbb{J}^{'}$, when the precedence graph is reversed, its image $j^R_{\mathbb{J}'\mathbb{J}^1}$ becomes an initial job in G^R. We have $h(j^R_{\mathbb{J}'\mathbb{J}^1}) > h(j^R_{\mathbb{J}'})$.

$j^R_{\mathbb{J}'\mathbb{J}^1}$ have priority over $j^R_{\mathbb{J}'}$ according to Hu's algorithm. As $j^R_{\mathbb{J}'}$ is executed in $\mathbb{J}^{'}$, which is the first time slot of (S_1), and as $j^R_{\mathbb{J}'\mathbb{J}^1}$ is also initial, we deduce that $j^R_{\mathbb{J}'\mathbb{J}^1}$ is in the first slot of (S_1) too, i.e. $j_{\mathbb{J}'\mathbb{J}^1}$ is in $\mathbb{J}^{'}$.

Case 2: $j_{\mathbb{J}'\mathbb{J}^1}$ has a successor in $\mathbb{J}^1$. This case does not actually exist, we show it by a contradiction. Let $j^{fs}_{\mathbb{J}'\mathbb{J}^1}$ be its last successor in $\mathbb{J}^1$. As $l(j^{fs}_{\mathbb{J}'\mathbb{J}^1}) < l(j_{\mathbb{J}'\mathbb{J}^1}) < l(j_{\mathbb{J}'})$, we find that $j^{fs}_{\mathbb{J}'\mathbb{J}^1}$ is in Case 1 we just proved. So, $j^{fs}_{\mathbb{J}'\mathbb{J}^1} \in \mathbb{J}^{'}$. It is a contradiction to the fact that $j^{fs}_{\mathbb{J}'\mathbb{J}^1}$ is in $\mathbb{J}^1$.

□

Corollary 14. *$\forall j_{\mathbb{J}'} \in \mathbb{J}^{'}$, $\forall j_{\mathbb{J}^1} \in \mathbb{J}^1$, if $l(j_{\mathbb{J}'}) > l(j_{\mathbb{J}^1})$, then $j_{\mathbb{J}^1}$ has a successor in $\mathbb{J}^{'}$.*

Propriety 3. *If $\mathbb{A}^{'} \neq \emptyset$, $\forall j_{\mathbb{A}'} \in \mathbb{A}^{'}$, $\forall j_{\mathbb{J}'\mathbb{J}^1} \in \mathbb{J}^1 \cup \mathbb{J}^{'}$ if $l(j_{\mathbb{A}'}) \geq l(j_{\mathbb{J}'\mathbb{J}^1})$ and $j_{\mathbb{J}'\mathbb{J}^1}$ has a direct successor in $\mathbb{J}^*$ then $j_{\mathbb{J}'\mathbb{J}^1} \in \mathbb{J}^{'}$.*

Proof. $j_{\mathbb{J}'\mathbb{J}^1}$ has a direct successor in $\mathbb{J}^*$ means that $j_{\mathbb{J}'\mathbb{J}^1}$ does not have a successor in $\mathbb{J}^{'}$ because every job has only one successor by definition of in-tree. So, their images $j^R_{\mathbb{A}'}$ and $j^R_{\mathbb{J}'\mathbb{J}^1}$ are initial jobs in G^R, the reverse graph.

Case 1 $l(j_{\mathbb{A}'}) > l(j_{\mathbb{J}'\mathbb{J}^1})$

Propriety 2 holds. $j_{\mathbb{J}'\mathbb{J}^1} \in \mathbb{J}^{'}$.

Case 2 $l(j_{\mathbb{A}'}) = l(j_{\mathbb{J}'\mathbb{J}^1})$

As $j_{\mathbb{J}'\mathbb{J}^1}$ has a direct successor in $\mathbb{J}^*$, and by definition $j_{\mathbb{A}'}$ does not, according to Step 2, $j^{'}_{\mathbb{J}'\mathbb{J}^1}$ takes priority over $j^{'}_{\mathbb{A}'}$ and finishes no later than $j^{'}_{\mathbb{A}'}$. So, $j_{\mathbb{J}'\mathbb{J}^1} \in \mathbb{J}^{'}$.

□

Propriety 4. *If $\mathbb{A}^* \neq \emptyset$ and $\mathbb{A}^{'} \neq \emptyset$, jobs in $\mathbb{A}^*$ and $\mathbb{A}^{'}$ are on the same level.*

Proof. First, prove that all jobs in $\mathbb{A}^*$ are on the same level.

Suppose by absurd that $\exists j^1_{\mathbb{A}^*}, j^2_{\mathbb{A}^*} \in \mathbb{A}^*$ such that $l(j^1_{\mathbb{A}^*}) \neq l(j^2_{\mathbb{A}^*})$, w.l.o.g. we assume $l(j^1_{\mathbb{A}^*}) < l(j^2_{\mathbb{A}^*})$. Because of the definition of level, we can find a direct predecessor of $j^1_{\mathbb{A}^*}$ on level $l(j^1_{\mathbb{A}^*})+1$, let it be $j^{1dp}_{\mathbb{A}^*}$. So, we have $l(j^{1dp}_{\mathbb{A}^*}) \leq l(j^2_{\mathbb{A}^*})$. $\forall j_{\mathbb{A}'} \in \mathbb{A}^{'}$, according to Propriety 1, $l(j^2_{\mathbb{A}^*}) \leq l(j_{\mathbb{A}'})$, so $l(j^{1dp}_{\mathbb{A}^*}) \leq l(j_{\mathbb{A}'})$. As $j^{1dp}_{\mathbb{A}^*}$ has a successor in $\mathbb{J}^*$, according to Propriety 3, we have $j^{1dp}_{\mathbb{A}^*} \in \mathbb{J}^{'}$, which is a contradiction to definition of $\mathbb{A}^*$. So, all jobs in $\mathbb{A}^*$ are on the same level. Now, we show that $\forall j_{\mathbb{A}'} \in \mathbb{A}^{'}$, $\forall j_{\mathbb{A}^*} \in \mathbb{A}^*$, $l(j_{\mathbb{A}^*}) = l(j_{\mathbb{A}'})$ to finish the proof.

Let $j^{lp}_{\mathbb{A}^*}$ be a lowest predecessor of $j_{\mathbb{A}^*}$ on level $l(j_{\mathbb{A}^*})+1$.

If $l(j^{lp}_{\mathbb{A}^*}) \leq l(j_{\mathbb{A}'})$, as $j^{lp}_{\mathbb{A}^*}$ has a successor in $\mathbb{J}^*$, according to Propriety 3, we have $j^{lp}_{\mathbb{A}^*} \in \mathbb{J}^{'}$, which is a contradiction to definition of $\mathbb{A}^*$. So, we have $l(j^{lp}_{\mathbb{A}^*}) > l(j_{\mathbb{A}'})$. So, $l(j_{\mathbb{A}'}) < l(j_{\mathbb{A}^*})+1$, which is $l(j_{\mathbb{A}'}) \leq l(j_{\mathbb{A}^*})$. By Propriety 1, we have $l(j_{\mathbb{A}'}) \geq l(j_{\mathbb{A}^*})$. So, $l(j_{\mathbb{A}'}) = l(j_{\mathbb{A}^*})$. □

Propriety 5. *Let $\mathbb{X}$ be a set of jobs on level $l^{\mathbb{X}}$, if either $\mathbb{X} \cap (\mathbb{J}^{'} \cup \mathbb{J}^1) = \emptyset$ or $\mathbb{X} \cap (\mathbb{A}^* \cup \mathbb{J}^1 \cup \mathbb{B}^{'}) = \emptyset$, then we have $|\mathbb{X}| \leq m$.*

Proof. We first consider the case when $\mathbb{X} \cap (\mathbb{J}^{'} \cup \mathbb{J}^1) = \emptyset$.

We build a partition of $\mathbb{X}$: $\mathbb{X}_{\mathbb{J}^*}$ and $\mathbb{X}_{\mathbb{J}^2}$, which represent the subset of $\mathbb{X}$ resp. in $\mathbb{J}^*$ and $\mathbb{J}^2$. As all jobs in $\mathbb{X}_{\mathbb{J}^2}$ have predecessors in $\mathbb{J}^*$, define these predecessors as $\mathbb{X}_{\mathbb{J}^*}^{\mathbb{J}^2}$. Then, $\forall j_{\mathbb{X}_{\mathbb{J}^*}^{\mathbb{J}^2}} \in \mathbb{X}_{\mathbb{J}^*}^{\mathbb{J}^2}$, we have $l(j_{\mathbb{X}_{\mathbb{J}^*}^{\mathbb{J}^2}}) > l^{\mathbb{X}}$. So, $\mathbb{X}_{\mathbb{J}^*}^{\mathbb{J}^2}$ has no intersection with $\mathbb{X}_{\mathbb{J}^*}$.

In an in-tree, different jobs on the same level have different predecessors, we have $|\mathbb{X}_{\mathbb{J}^*}^{\mathbb{J}^2}| \geq |\mathbb{X}_{\mathbb{J}^2}|$, so $|\mathbb{X}| = |\mathbb{X}_{\mathbb{J}^*}| + |\mathbb{X}_{\mathbb{J}^2}| \leq |\mathbb{X}_{\mathbb{J}^*}| + |\mathbb{X}_{\mathbb{J}^*}^{\mathbb{J}^2}| < m$, which closes the first case.

Now, we consider the second case, where $\mathbb{X} \cap (\mathbb{A}^* \cup \mathbb{J}^1 \cup \mathbb{B}^{'}) = \emptyset$.

Partition $\mathbb{X}$ into $\mathbb{X}_{\mathbb{A}^{'}}$, $\mathbb{X}_{\mathbb{B}^*}$, and $\mathbb{X}_{\mathbb{J}^2}$, which represent resp. subset of $\mathbb{X}$ in $\mathbb{A}^{'}$, $\mathbb{B}^*$ and $\mathbb{J}^2$. Similarly, we define the predecessors of $\mathbb{X}_{\mathbb{J}^2}$ in $\mathbb{A}^*$ and $\mathbb{B}^*$ as $\mathbb{X}_{\mathbb{A}^*}^{\mathbb{J}^2}$ and $\mathbb{X}_{\mathbb{B}^*}^{\mathbb{J}^2}$, we have $|\mathbb{X}_{\mathbb{A}^*}^{\mathbb{J}^2}| + |\mathbb{X}_{\mathbb{B}^*}^{\mathbb{J}^2}| \geq |\mathbb{X}_{\mathbb{J}^2}|$.

$\mathbb{X}_{\mathbb{A}^{'}}$, $\mathbb{X}_{\mathbb{B}^*}$, $\mathbb{X}_{\mathbb{A}^*}^{\mathbb{J}^2}$ and $\mathbb{X}_{\mathbb{B}^*}^{\mathbb{J}^2}$ have no intersection with each other because jobs in $\mathbb{X}_{\mathbb{A}^*}^{\mathbb{J}^2}$ and $\mathbb{X}_{\mathbb{B}^*}^{\mathbb{J}^2}$ are not on level $l^{\mathbb{X}}$.

Case 1 If $\mathbb{X}_{\mathbb{A}^*}^{\mathbb{J}^2} \neq \emptyset$:

If $\mathbb{A}^{'} = \emptyset$, $\mathbb{X}_{\mathbb{A}^{'}} = \emptyset$; if $\mathbb{A}^{'} \neq \emptyset$, we still have $\mathbb{X}_{\mathbb{A}^{'}} = \emptyset$ because applying Propriety 4 leads to know that jobs in $\mathbb{X}_{\mathbb{A}^{'}}$ would be on the same level as jobs in $\mathbb{X}_{\mathbb{A}^*}^{\mathbb{J}^2}$, which is, however, strictly higher than $l^{\mathbb{X}}$. As $\mathbb{X}_{\mathbb{B}^*}$, $\mathbb{X}_{\mathbb{A}^*}^{\mathbb{J}^2}$ and $\mathbb{X}_{\mathbb{B}^*}^{\mathbb{J}^2}$ are subset of $\mathbb{J}^*$, we have:

$$|\mathbb{X}| \leq |\mathbb{X}_{\mathbb{A}^*}^{\mathbb{J}^2}| + |\mathbb{X}_{\mathbb{B}^*}^{\mathbb{J}^2}| + |\mathbb{X}_{\mathbb{B}^*}| \leq |\mathbb{J}^*| < m$$

Case 2 If $\mathbb{X}_{\mathbb{A}^*}^{\mathbb{J}^2} = \emptyset$:

Define $\mathbb{X}^{\mathbb{B}^{'}}$ the predecessors of $\mathbb{X}_{\mathbb{B}^*}$ and $\mathbb{X}_{\mathbb{B}^*}^{\mathbb{J}^2}$ in $\mathbb{B}^{'}$.

As jobs in $\mathbb{X}^{\mathbb{B}^{'}}$ are not on level $l^{\mathbb{X}}$, we have $\mathbb{X}^{\mathbb{B}^{'}} \cap \mathbb{X} = \emptyset$. As subsets of $\mathbb{J}^{'}$, we have:

$$|\mathbb{X}^{\mathbb{B}^{'}}| + |\mathbb{X}_{\mathbb{A}^{'}}| \leq |\mathbb{J}^{'}| = m$$

As $|\mathbb{X}^{\mathbb{B}^{'}}| \geq |\mathbb{X}_{\mathbb{B}^*}^{\mathbb{J}^2}| + |\mathbb{X}_{\mathbb{B}^*}| \geq |\mathbb{X}_{\mathbb{J}^2}| + |\mathbb{X}_{\mathbb{B}^*}|$, we have

$$|\mathbb{X}| = |\mathbb{X}_{\mathbb{A}^{'}}| + |\mathbb{X}_{\mathbb{J}^2}| + |\mathbb{X}_{\mathbb{B}^*}| \leq m$$

□

Propriety 6. *$\forall \mathbb{X} \subset \mathbb{J}^T$, $\forall j \in \mathbb{X}$, jobs in $\mathbb{S}_{\mathbb{X}}(j)$ have no successor in $\mathbb{X}$.*

Proof. Suppose by absurd that $j_{\mathbb{S}} \in \mathbb{S}_{\mathbb{X}}(j)$ has a successor $j_{\mathbb{S}}^s$ in $\mathbb{X}$. As $l(j_{\mathbb{S}}^s) < l(j_{\mathbb{S}})$, $l(j_{\mathbb{S}}^s) < l(j)$, and we have $j_{\mathbb{S}}^s \in \mathbb{L}_{\mathbb{X}}(j)$, which is a contradiction to definition of $\mathbb{S}_{\mathbb{X}}(j)$. □

Corollary 15. Let $\mathbb{X}$ be any superset of $\mathbb{B}^*$, jobs in $\mathbb{S}_{\mathbb{X}}(j)$ neither are in $\mathbb{B}^{'}$ nor have successor in $\mathbb{B}^{'}$.

Proof. $\forall j_{\mathbb{S}} \in \mathbb{S}_{\mathbb{X}}(j)$, whether $j_{\mathbb{S}} \in \mathbb{B}^{'}$ or $j_{\mathbb{S}}$ has a successor in $\mathbb{B}^{'}$, can we find a successor of $j_{\mathbb{S}}$ in $\mathbb{B}^*$, which is also in $\mathbb{X}$. This contradicts Propriety 6. □

Propriety 7. *If $\mathbb{A}^* \neq \emptyset$ and $\mathbb{B}^* \neq \emptyset$, then $\forall j_{\mathbb{A}^*} \in \mathbb{A}^*$, $\forall j_{\mathbb{B}^*} \in \mathbb{B}^*$, we have $l(j_{\mathbb{A}^*}) \geq l(j_{\mathbb{B}^*})$*

Proof. Suppose by absurd that $l(j_{\mathbb{A}^*}) < l(j_{\mathbb{B}^*})$. We can find a lowest direct predecessor $j_{\mathbb{A}^*}^{ldp}$ of $j_{\mathbb{A}^*}$, and a predecessor $j_{\mathbb{B}^*}^{p}$ of $j_{\mathbb{B}^*}$ in $\mathbb{B}^{'}$. We have $l(j_{\mathbb{B}^*}^{p}) > l(j_{\mathbb{B}^*}) \geq l(j_{\mathbb{A}^*}^{ldp})$. As the only direct successor of $j_{\mathbb{A}^*}^{ldp}$ is in $\mathbb{J}^*$, $j_{\mathbb{A}^*}^{ldp}$ cannot have any successor in $\mathbb{J}^{'}$. Using Propriety 2, $j_{\mathbb{A}^*}^{ldp} \in \mathbb{J}^{'}$, which is a contradiction to definition of $\mathbb{A}^*$. □

Propriety 8. *$\forall j_{\mathbb{A}^*} \in \mathbb{A}^*$, each job in $\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{A}^*}) \cap \mathbb{J}^2$ has a predecessor in $\mathbb{A}^*$.*

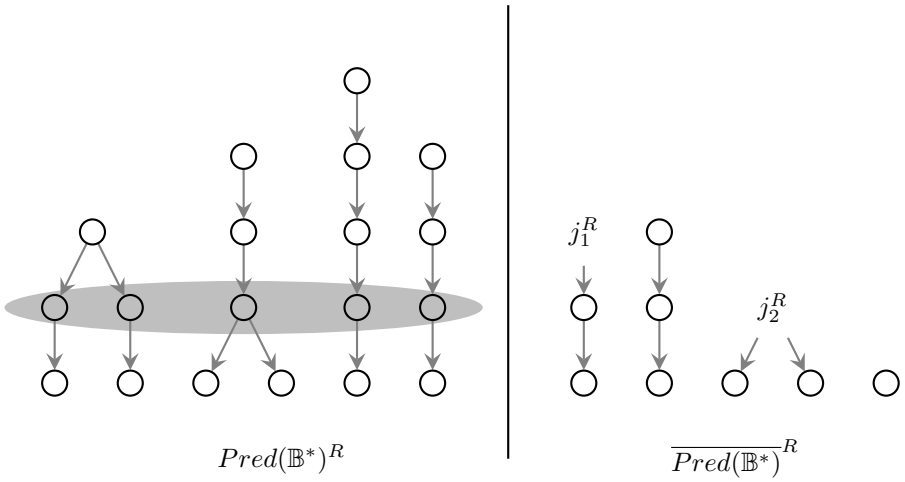

Figure 4.7: G^R: when executing j_1^R, the available jobs in $pred(\mathbb{B}^*)^R$ prior to j_2^R are in the oval.

Proof. Each job in $\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{A}^*}) \cap \mathbb{J}^2$ has a predecessor in $\mathbb{J}^*$. This predecessor is strictly higher than $j_{\mathbb{A}^*}$, and according to Propriety 7, jobs in $\mathbb{B}^*$ are not higher than $j_{\mathbb{A}^*}$. So, this predecessor can only be in $\mathbb{A}^*$. □

Propriety 9. *If $\mathbb{A}^{'} = \emptyset$, $\forall j_1, j_2$ two final jobs in $\mathbb{J}^1 \setminus pred(\mathbb{B}^*)$, such that $l(j_1) < l(j_2)$, then $C_{j_1} > C_{j_2}$.*

Proof. According to the fact that $\mathbb{J}^1 \cup \mathbb{J}^{'}$ is scheduled by reversing the result of Hu's algorithm of G^R. Let j_1^R, j_2^R be the images of j_1, j_2 in G^R. As j_1^R, j_2^R are initial jobs, by (4.1), $h(j_1^R) > h(j_2^R)$, and according to Hu's algorithm, we have $C_{j_1^R} \leq C_{j_2^R}$. We now prove that they are not in the same time slot, i.e. $C_{j_1^R} \neq C_{j_2^R}$.

We separate $\mathbb{J}^1 \cup \mathbb{J}^{'}$ into two parts: $pred(\mathbb{B}^*)$ and $\overline{pred(\mathbb{B}^*)} =: \mathbb{J}^1 \cup \mathbb{J}^{'} \setminus pred(\mathbb{B}^*)$. Then, note their images as $pred(\mathbb{B}^*)^R$ and $\overline{pred(\mathbb{B}^*)}^R$. As $\mathbb{A}^{'} = \emptyset$, $pred(\mathbb{B}^*)^R$ contains at least m initial jobs. Any finished job in $pred(\mathbb{B}^*)^R$ can release at least one successor by the structure out-tree of G^R. As $pred(\mathbb{B}^*)^R$ are higher than $\overline{pred(\mathbb{B}^*)}^R$, when it comes to the execution of j_1^R, at least m jobs in $pred(\mathbb{B}^*)^R$ with priority to j_2^R are available, see Fig. 4.7. They fill this slot, and j_2^R has to be executed later. So, $C_{j_1^R} < C_{j_2^R}$, i.e. $C_{j_1} > C_{j_2}$. □

Propriety 10. *If $\mathbb{A}^* \neq \emptyset$, $\forall j_{\mathbb{B}^*} \in \mathbb{B}^*$, then $|\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{B}^*})| \leq 2m$. Furthermore, if $l(j_{\mathbb{B}^*}) < l(j_{\mathbb{A}^*}^{lowest})$, where $j_{\mathbb{A}^*}^{lowest}$ is a lowest job in $\mathbb{A}^*$, then $|\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{B}^*})| \leq m$.*

Proof. We first prove $\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{B}^*}) \cap \mathbb{J}^1 = \emptyset$. Suppose by absurd that $\exists j_{\mathbb{S}} \in \mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{B}^*}) \cap \mathbb{J}^1$. By Corollary 14, $j_{\mathbb{S}}$ has a successor in $\mathbb{J}^{'}$, let it be $j_{\mathbb{S}}^s$. By Corollary 15, $j_{\mathbb{S}}^s \notin \mathbb{B}^{'}$, so $j_{\mathbb{S}}^s \in \mathbb{A}^{'}$. As Propriety 4 and 7 hold, we have $l(j_{\mathbb{S}}^s) \geq l(j_{\mathbb{B}^*}^h)$, where $j_{\mathbb{B}^*}^h$ is the highest job in $\mathbb{B}^*$. However, $j_{\mathbb{S}}^s$ is a successor of $j_{\mathbb{S}}$, $l(j_{\mathbb{S}}) = l(j_{\mathbb{B}^*}) > l(j_{\mathbb{S}}^s) \geq l(j_{\mathbb{B}^*}^h)$, contradicts the fact that $j_{\mathbb{B}^*}^h$ is the highest job in $\mathbb{B}^*$.

Then, by Corollary 15, we have $\mathbb{S}_{\mathbb{B}^*}(j_{\mathbb{B}^*}) \cap (\mathbb{J}^1 \cup \mathbb{B}^{'}) = \emptyset$.

Partition $\mathbb{S}_{\mathbb{B}^*}(j_{\mathbb{B}^*})$ into $\mathbb{S}_{\mathbb{B}^*}(j_{\mathbb{B}^*}) \setminus \mathbb{A}^*$ and $\mathbb{S}_{\mathbb{B}^*}(j_{\mathbb{B}^*}) \cap \mathbb{A}^*$. According to Propriety 5, $|\mathbb{S}_{\mathbb{B}^*}(j_{\mathbb{B}^*}) \setminus \mathbb{A}^*| \leq m$. We also have $|\mathbb{S}_{\mathbb{B}^*}(j_{\mathbb{B}^*}) \cap \mathbb{A}^*| \leq |\mathbb{A}^*| < m$. So, $|\mathbb{S}_{\mathbb{B}^*}(j_{\mathbb{B}^*})| = |\mathbb{S}_{\mathbb{B}^*}(j_{\mathbb{B}^*}) \setminus \mathbb{A}^*| + |\mathbb{S}_{\mathbb{B}^*}(j_{\mathbb{B}^*}) \cap \mathbb{A}^*| < 2m$.

In particular, when $l(j_{\mathbb{B}^*}) < l(j_{\mathbb{A}^*}^{lowest})$, $\mathbb{S}_{\mathbb{B}^*}(j_{\mathbb{B}^*}) \cap \mathbb{A}^* = \emptyset$. In this case, $|\mathbb{S}_{\mathbb{B}^*}(j_{\mathbb{B}^*})| \leq m$. □

Propriety 11. *If $\mathbb{A}^{'} = \emptyset$, $\forall j_{\mathbb{A}^*} \in \mathbb{A}^*$ such that $\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{A}^*}) \cap \mathbb{J}^1 \neq \emptyset$ then*

1. *$j_{\mathbb{A}^*}$ is a highest job in $\mathbb{A}^*$ and jobs in $\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{A}^*}) \cap \mathbb{J}^1$ are final.*
2. *$\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{A}^*}) \setminus \mathbb{J}^* \subset \mathbb{J}^1$.*

Proof. First, let $j^h_{\mathbb{A}^*}$ be a highest job of $\mathbb{A}^*$ and its last finished predecessor be $j^{lp}_{\mathbb{A}^*}$. $\forall j_{\mathbb{S}} \in \mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{A}^*}) \cap \mathbb{J}^1$, consider its last successor $j^{fs}_{\mathbb{S}}$ in $\mathbb{J}^1$, if $j_{\mathbb{S}}$ is final then $j^{fs}_{\mathbb{S}} = j_{\mathbb{S}}$. We have

$$l(j^{fs}_{\mathbb{S}}) \leq l(j_{\mathbb{S}}) = l(j_{\mathbb{A}^*}) \leq l(j^h_{\mathbb{A}^*}) < l(j^{lp}_{\mathbb{A}^*})$$

In Step 5 when constructing (S), we exchange jobs when the 3 conditions are satisfied. Now, check the following conditions of switching $j^{fs}_{\mathbb{S}}$ and $j^h_{\mathbb{A}^*}$:

1. by Propriety 9, $j^{lp}_{\mathbb{A}^*}$ is finished strictly before $j^{fs}_{\mathbb{S}}$;
2. by Propriety 6, $j_{\mathbb{S}}$ has no successor in $\mathbb{J}^*$, which implies that $j^{fs}_{\mathbb{S}}$ is final and has no successor in $\mathbb{J}^*$.

If $l(j^{fs}_{\mathbb{S}}) < l(j^h_{\mathbb{A}^*})$, $j^{fs}_{\mathbb{S}}$ should have been switched with $j^h_{\mathbb{A}^*}$ by Step 5. So, we have $l(j^{fs}_{\mathbb{S}}) = l(j^h_{\mathbb{A}^*})$ and the inequality above becomes

$$l(j^{fs}_{\mathbb{S}}) = l(j_{\mathbb{S}}) = l(j_{\mathbb{A}^*}) = l(j^h_{\mathbb{A}^*}) < l(j^{lp}_{\mathbb{A}^*})$$

This implies $j_{\mathbb{S}} = j^{fs}_{\mathbb{S}}$, i.e. $j_{\mathbb{S}}$ is final; and $l(j_{\mathbb{A}^*}) = l(j^h_{\mathbb{A}^*})$, i.e. $j_{\mathbb{A}^*}$ is a highest job.

Now, we prove the $\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{A}^*}) \setminus \mathbb{J}^* \subset \mathbb{J}^1$. Job in $\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{A}^*}) \cap \mathbb{J}^2$ would have predecessor in $\mathbb{A}^*$ by Propriety 8, which contradicts the fact that $j_{\mathbb{A}^*}$ is already a highest job in $\mathbb{A}^*$, so $\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{A}^*}) \cap \mathbb{J}^2 = \emptyset$. Then, $\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{A}^*}) \cap \mathbb{J}^{'} = \emptyset$ can be deduced by Corollary 15. While we have both $\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{A}^*}) \cap \mathbb{J}^2 = \emptyset$ and $\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{A}^*}) \cap \mathbb{J}^{'} = \emptyset$, we have $\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{A}^*}) \setminus \mathbb{J}^* \subset \mathbb{J}^1$.

□

4.2.4 Proof for the regularity

Using all the previous properties of (S), we prove the regularity of $\mathbb{J}^*$.

Case 1: $\mathbb{A}^* = \emptyset$

We assume that $\mathbb{A}^* = \emptyset$. We prove that in this case, $\mathbb{J}^*$, which is $\mathbb{B}^*$, has a pure-m-limited superset.

Let us define such a superset as $\mathbb{B}^*$ unions the jobs in $\mathbb{A}^{'}$ which are strictly lower than jobs in $\mathbb{B}^*$:

$$\mathbb{V} \coloneqq \{j_{\mathbb{A}'} \in \mathbb{A}^{'} | \forall j_{\mathbb{B}^*} \in \mathbb{B}^*, l(j_{\mathbb{A}'}) < l(j_{\mathbb{B}^*})\} \cup \mathbb{B}^*$$

We prove that $\mathbb{V}$ is pure-m-limited: $\forall j \in \mathbb{V}$, $|\mathbb{S}_{\mathbb{V}}(j)| \leq m$. Consider $\forall j_{\mathbb{S}} \in \mathbb{S}_{\mathbb{V}}(j)$, we show that $j_{\mathbb{S}} \notin \mathbb{J}^1$.

We find $j^h_{\mathbb{B}'}$ the highest job in $\mathbb{B}^{'}$, and we have $l(j_{\mathbb{S}}) < l(j^h_{\mathbb{B}'})$. If $j_{\mathbb{S}} \in \mathbb{J}^1$, according to Corollary 14, $j_{\mathbb{S}}$ has a successor in $\mathbb{J}^{'}$, let it be $j^s_{\mathbb{S}}$. However, $j^s_{\mathbb{S}} \in \mathbb{A}^{'}$ implies that $j^s_{\mathbb{S}} \in \mathbb{V}$ because $l(j^s_{\mathbb{S}}) < l(j_{\mathbb{S}})$; $j^s_{\mathbb{S}} \in \mathbb{B}^{'}$ implies that $j^s_{\mathbb{S}}$ has successor in $\mathbb{B}^*$. In both case can we find a successor of $j_{\mathbb{S}}$ in $\mathbb{V}$, which contradicts Propriety 6.

Corollary 15 indicates that $\mathbb{S}_{\mathbb{V}}(j) \cap \mathbb{B}^{'} = \emptyset$, so, $\mathbb{S}_{\mathbb{V}}(j) \cap (\mathbb{B}^{'} \cup \mathbb{J}^1) = \emptyset$. By Propriety 5, as $\mathbb{A}^* = \emptyset$, we have $|\mathbb{S}_{\mathbb{V}}(j)| \leq m$.

As we proved that $\mathbb{V}$ is pure-m-limited, $\mathbb{J}^*$ is regular when $\mathbb{A}^* = \emptyset$.

Case 2: $\mathbb{A}^* \neq \emptyset$

The rest of our proof shows that when $\mathbb{A}^* \neq \emptyset$, jobs in $\mathbb{J}^*$ are in one of the 3 groups: Group 1, Group 2, and Group 3.

Firstly, consider jobs in $\mathbb{B}^*$. $\forall j_{\mathbb{B}^*} \in \mathbb{B}^*$, we prove that $j_{\mathbb{B}^*}$ is in either Group 1 or Group 2.

Proof. By Propriety 10, if $l(j_{\mathbb{B}^*}) < l(j^{lowest}_{\mathbb{A}^*})$, then $j_{\mathbb{B}^*}$ is m-limited. Otherwise, $j_{\mathbb{B}^*}$ is $2m$-limited. Jobs in Group 2 are the highest jobs of $\mathbb{B}^*$, on the same level. □

Secondly, let us consider $\mathbb{A}^*$ in the subcase where $\mathbb{A}^{'} \neq \emptyset$. $\forall j_{\mathbb{A}^*} \in \mathbb{A}^*$, we prove that $j_{\mathbb{A}^*}$ is in Group 3.

Proof. Suppose by absurd that $j_{\mathbb{S}}$ is the lexicographically first job in $\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{A}^*})$, $j_{\mathbb{S}} \neq j_{\mathbb{A}^*}$, and $j_{\mathbb{S}} \notin \mathbb{J}^*$. If $j_{\mathbb{S}} \in \mathbb{J}^2$, according to Propriety 8, it has a predecessor in $\mathbb{A}^*$. However, this contradicts the fact that jobs in $\mathbb{A}^*$ are on the same level according to Propriety 4. So, $j_{\mathbb{S}} \in \mathbb{J}^1 \cup \mathbb{J}^{'}$.

As $\mathbb{A}^{'} \neq \emptyset$, Propriety 4 also tells that jobs in $\mathbb{A}^{'}$ is on the same level as $j_{\mathbb{S}}$. This means during Hu's algorithm in Step 2 when constructing (S), $j_{\mathbb{S}}$ has the same priority as jobs in $\mathbb{A}^{'}$. As we execute jobs in

lexicographic order and $j_{\mathbb{S}}$ is the lexicographically first one, we have $j_{\mathbb{S}} \in \mathbb{J}^{'}$. By Corollary 15, $j_{\mathbb{S}} \notin \mathbb{B}^{'}$. So, $j_{\mathbb{S}} \in \mathbb{A}^{'}$.

However, in Step 5 when constructing (S), we exchange jobs when 3 conditions are satisfied. Now, check the following 3 conditions:

1. As $j_{\mathbb{S}} \in \mathbb{A}^{'}$, $succ(j_{\mathbb{S}})$ begins strictly after $\mathbb{J}^*$.
2. As $j_{\mathbb{A}^*} \in \mathbb{A}^*$, $prec(j_{\mathbb{A}^*})$ finishes strictly before $\mathbb{J}^{'}$.
3. $l(j_{\mathbb{S}}) = l(j_{\mathbb{A}^*})$, and $j_{\mathbb{S}}$ lexicographically precedes $j_{\mathbb{A}^*}$.

$j_{\mathbb{S}}$ should have been switched with $j_{\mathbb{A}^*}$ by Step 5. So, we have a contradiction. As $j_{\mathbb{A}^*}$ is the lexicographically first one, it is in Group 3. □

Thirdly, we consider the other subcase, i.e. $\mathbb{A}^{'} = \emptyset$. $\forall j_{\mathbb{A}^*} \in \mathbb{A}^*$, we prove that $j_{\mathbb{A}^*}$ is in either Group 1 or Group 3.

Proof. If $j_{\mathbb{A}^*}$ is not a highest job in $\mathbb{A}^*$, we show that it is in Group 1. By Corollary 15 and Propriety 11, $\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{A}^*}) \cap (\mathbb{B}^{'} \cup \mathbb{J}^1) = \emptyset$, as $\mathbb{B}^{'} = \mathbb{J}^{'}$, by Propriety 5, $|\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{A}^*})| < m$.

If $j_{\mathbb{A}^*}$ is a highest job, we show that it is in Group 3. Suppose by absurd that $j_{\mathbb{S}}$ is the lexicographically first job in $\mathbb{S}_{\mathbb{J}^*}(j_{\mathbb{A}^*})$, $j_{\mathbb{S}} \neq j_{\mathbb{A}^*}$, and $j_{\mathbb{S}} \notin \mathbb{J}^*$. By Propriety 11, we have $j_{\mathbb{S}} \in \mathbb{J}^1$.

In Step 5 when constructing (S), we exchange jobs when 3 conditions are satisfied. Now, we can state that:

1. The predecessors of $j_{\mathbb{A}^*}$, which are higher than $j_{\mathbb{S}}$, finish before $j_{\mathbb{S}}$ according to Propriety 9.
2. By Propriety 11, $j_{\mathbb{S}}$ is final
3. $l(j_{\mathbb{S}}) = l(j_{\mathbb{A}^*})$, and $j_{\mathbb{S}}$ lexicographically precedes $j_{\mathbb{A}^*}$.

$j_{\mathbb{S}}$ should have been switched with $j_{\mathbb{A}^*}$ in Step 5. So, we have a contradiction. □

Propriety 7 reveals that Group 3 which contains jobs in $\mathbb{A}^*$ are not lower than Group 2, which contains jobs in $\mathbb{B}^*$. Thus, jobs in $\mathbb{A}^*$ and $\mathbb{B}^*$ are in one of the 3 groups, we accomplished the proof of regularity of $\mathbb{J}^*$.

4.3 The algorithm

We just proved that there always exists a regular set $\mathbb{J}^*$. To find the regular $\mathbb{J}^*$, we only need to explore all regular sets, which are recursively listed by Algorithm 1.

The sets are generated by choosing jobs from bottom to the top. Jobs added to $\mathfrak{C}$ by line 10, 16 or 20 correspond to the 3 groups:

1. A m-limited job in Group 1 is picked from at most hm candidates.
2. A $2m$-limited job in Group 2 is picked from at most $2hm$ candidates.
3. The lexicographically first job in Group 3 is picked from at most h candidates.

As each job is picked from at most $2hm$ candidates, noticing that m is fixed, the complexity is $O(h^m)$. Line 26 is dedicated to the case where $\mathbb{J}^*$ has a pure-m-limited superset. Creating subsets from at most $m-1$ jobs is in $O(2^m)$, which is a constant when m is fixed. Thus, the total complexity of the algorithm is $O(h^m)$.

4.3.1 Example

We illustrate how the algorithm lists regular sets by the example in Fig. 4.8 where 19 jobs are to be scheduled by $m = 3$ parallel machines.

When $k = 0$ and $\mathbb{L} = \emptyset$, the first loop gives:

- Line 10 adds $\{16\}, \{17\}, \{18\}, \{19\}, \{16, 17\}$
- line 16 adds $\{1\}, \{12\}$

Input: set of all jobs $\mathbb{J}^T$
Output: $\mathfrak{C}$: collection of regular sets

Add $\emptyset$ to $\mathfrak{C}$;
// Partitionable into 3 groups
for $k = 0, 1, \ldots, m-2$ **do**
  foreach $\mathbb{L} \in \mathfrak{C}$, *s.t.* $|\mathbb{L}| = k$ **do**
    $\mathbb{P} = \{j_1 \in \mathbb{J}^T | l(j_1) > l(j_2), \forall j_2 \in \mathbb{L}\} \setminus pred(\mathbb{L})$;
    foreach *level* l *from* 0 *to* h **do**
      Let $\mathbb{S} = \{j \in \mathbb{P} | l(j) = l\}$;
      // Group 1
      if $|\mathbb{S}| \leq m$ **then**
        foreach *subset* $\mathbb{S}' \subset \mathbb{S}$ **do**
          if $|\mathbb{S}' \cup \mathbb{L}| \leq m-1$ **then**
            add $\mathbb{S}' \cup \mathbb{L}$ to $\mathfrak{C}$
          end
        end
      end
      // Group 2
      if $2m \geq |\mathbb{S}| > m$ *and* $\mathbb{L}$ *is pure-m-limited* **then**
        foreach *subset* $\mathbb{S}'$ *of* $\mathbb{S}$ **do**
          Add $\mathbb{S}' \cup \mathbb{L}$ to $\mathfrak{C}$;
        end
      end
      // Group 3
      Let j be the lexicographically first job of $\mathbb{S}$;
      Add $\{j\} \cup \mathbb{L}$ to $\mathfrak{C}$;
    end
  end
end
// pure-m-limited superset
foreach $\mathbb{L} \in \mathfrak{C}$ **do**
  if $\mathbb{L}$ *is pure-m-limited* **then**
    Add all subsets of $\mathbb{L}$ to $\mathfrak{C}$
  end
end

Algorithm 1: Get all regular sets

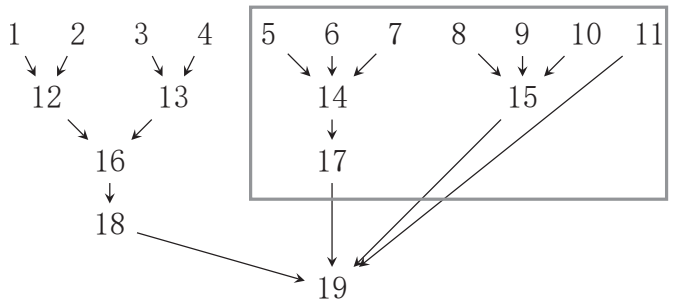

Figure 4.8: Example, where $\mathbb{P}$ is in the rectangle when $\mathbb{L} = \{16\}$

	B	N	ratio	B	N	ratio
n	$m = 3$			$m = 4$		
35	51.91	11.92	4.35	467.71	52.72	8.87
45	202.24	12.35	16.38	2193.58	90.64	24.20
55	550.41	13.58	40.52	7860.83	104.08	75.53
65	1513.29	92.84	16.30	22862.25	143.65	159.16
75	3249.82	75.71	42.92	62671.95	457.11	137.11
85	6751.65	98.98	68.22	132092.10	753.87	175.22
95	12653.46	310.94	40.69	277052.60	1253.26	221.07
105	22044.72	335.44	65.72	548726.40	2732.75	200.80
115	39132.76	476.69	82.09	1041885.00	3369.15	309.24

Table 4.1: Experimental results: average time in seconds, where B,N stand for Baptiste's algorithm and the new algorithm

- line 20 adds $\{13\}, \{14\}, \{15\}, \{12, 13\}$, $\{12, 14\}, \{12, 15\}, \{13, 14\}, \{13, 15\}, \{14, 15\}$.

When $k = 1$, several loops are done. Take an example of $\mathbb{L} = \{16\}$

- Line 10 adds $\{16, 14\}, \{16, 15\}, \{16, 17\}$
- line 16 adds $\{16, 5\}$
- line 20 does not add any set because no level contains 4, 5 or 6 jobs.

The algorithm runs similarly for other choice of $\mathbb{L}$. The final results are given by the following table:

k	pure-m-limited sets	impure-m-limited sets
k=0	$\{16\}, \{17\}, \{18\}, \{19\}, \{1\}$	$\{12\}, \{13\}, \{14\}, \{15\}$ $, \{12, 13\}, \{12, 14\}, \{12, 15\},$ $\{13, 14\}, \{13, 14\}, \{14, 15\},$
k=1	$\{16, 14\}, \{16, 15\}, \{16, 17\},$ $\{17, 12\}, \{17, 13\}, \{17, 15\},$ $\{18, 17\}, \{18, 14\}, \{18, 15\}$	$\{16, 5\}, \{17, 1\}, \{18, 5\},$ $\{12, 3\}, \{13, 1\}, \{14, 1\},$ $\{15, 1\}$

As all subsets of pure-m-limited sets are already in $\mathfrak{C}$, line 26 does not add any other sets.

Our algorithm returns $|\mathfrak{C}| = 31$ sets, which is much less than Baptiste's algorithm, who requires comparison of $\binom{19}{1} + \binom{19}{2} = 190$ candidates. The set $\mathbb{J}^*$ is $\{19\}$.

4.4 Experiments

Our algorithm generates all regular sets in time $O(h^m)$, while it requires extra calculation, for example, a tree-pruning of $\mathbb{P}$ in every step. It is still meaningful to implement our algorithm and compare it with Baptiste's algorithm on the running time.

For each couple n, m, we create 5 different instances with a height from 3 to 6, on which we ran both algorithms. The instances of in-trees are randomly generated in the following way:

1. generate a chain of length $h = 3, \ldots, 6$
2. create n jobs
3. randomly (equiprobably) assign each job a successor from candidate jobs where candidate jobs are those who have already at least 1 but at most $h - 1$ successors

We report the average time consumed in table 4.1.

As it can be seen, both algorithm consumed exponential time to n while our algorithm performs much better. The theoretical ratio is $\frac{O(n^m)}{O(h^m)} = O(w^m)$ where $w = \frac{n}{h}$ can be regarded as the *average width*. However, the experimental ratio is not so perfectly exponential to n due to the randomness.

4.5 Conclusion

We developed an algorithm for the scheduling problem $P|p_j = p, i.t.|\sum C_j$, with a complexity $O(h^m)$, where h is the height of in-tree if the number of machines is considered as a constant. This leads to the conclusion that the complexity is mainly determined by the height of the graph.

However, based on the properties we proved, this algorithm still has large room for improvement. For example, a more strict and efficient constraint $|\mathbb{L}| \leq m - |\mathbb{X}|$ can replace $|\mathbb{L}| \leq m$. Moreover, to make the algorithm practical for real instances, several technical improvements can be made, such as checking whether $|\mathbb{J}^1 \cup \mathbb{J}^{'}|$ is divisible by m before generating a schedule from every candidate set. These techniques may refine the algorithm to a certain extent, however, not in the sense of complexity.

Chapter 5

Modeling $P|prec|C_{\max}$

Considering the makespan, we recall that the problem is $\mathcal{NP}$-hard even without precedence constraints and a fixed number of machines equal to 2(Lenstra et al., 1977). In previous chapters, we discussed that even when the jobs have equal-processing-time, and the precedence constraints take particular shape, the problem is still $\mathcal{NP}$-hard for most of the cases. Nevertheless, generalized as a combinatorial optimization problem, all scheduling problems can be solved by various general problem-solving methods, such as tree-search procedures and mixed integer programming which benefits from its intuitive modeling and efficient solvers.

We are interested in this chapter in proposing a way to solve the general scheduling problem $P|prec|C_{\max}$ with arbitrary precedence constraints and arbitrary processing-time. This method will be adaptable for specific processing time or precedence graph.

We adapt some models, which are originally designed for other problems, to parallel scheduling problems and propose a new one. Then, we compare their performance by testing them on benchmarks with precedence constraints from (Kolisch and Sprecher, 1996).

5.1 Preliminary

Every model to be presented uses the variables S_j and C_j as starting time and completion time of j. The objective function is $C_{\max}$, the following constraints hold for all models, and they are omitted hereafter:

$$\begin{aligned} C_j &= S_j + p_j, \quad \forall j \in \mathbb{J}^T \\ C_{max} &\geq C_j, \quad \forall j \in \mathbb{J}^T \\ S_j &\geq C_{j'}, \quad \forall j^{'}, j \in \mathbb{J}^T \text{ and } j^{'} \prec j \end{aligned}$$

where p_j is the processing time of job j and $j^{'} \prec j$ means $j^{'}$ precedes j. $\mathbb{J}^T$ and $\mathbb{M}^T$ are the sets of all jobs and machines. In the following sections, $j^{'}, j \in \mathbb{J}^T$ and $\theta \in \mathbb{M}^T$. M represents a large number, which can be defined as $\sum p_j$.

Indeed, any one of C_j and S_j can be represented by the other one. However, the model is solved faster using both C_j and S_j then using just one of them, as we can see from the experimental results.

5.2 Time-Indexed Model (TIM)

The time-indexed model (TIM) is formulated in Thomalla (2001) for job shop scheduling problems. The principal variables it uses are $x^{\theta}_{t,j}$ which is 1 if j starts on θ at t. It requires an estimation of an upper bond of C_{max}, which is noticed as t_{max}. We proposed a version for TIM by using a nominal variable C_j:

$$S_{j'} \geq t x^{\theta}_{t,j'}, \quad \forall j^{'} \in \mathbb{J}^T, t \leq t_{max}; \theta \in \mathbb{M}^T \tag{5.1a}$$

$$x^{\theta_1}_{t_1,j'} + x^{\theta_2}_{t_2,j} \leq 1, \quad \forall j^{'} \prec j, t_1 + p_{j'} \geq t_2, \theta_1, \theta_2 \in \mathbb{M}^T \tag{5.1b}$$

$$\sum_{t \leq t_{max}} \sum_{\theta \in \mathbb{M}^T} x^{\theta}_{t,j'} = 1, \quad \forall j^{'} \in \mathbb{J}^T \tag{5.1c}$$

$$\sum_{j^{'} \in \mathbb{J}^T} x^{\theta}_{t,j^{'}} \leq 1, \quad \forall \theta \in \mathbb{M}^T, j^{'} \in \mathbb{J}^T \tag{5.1d}$$

(5.1a) defines $C_{j'}$. (5.1b) describes the precedence constraints. (5.1c) and (5.1d) make sure that each job is executed only once.

In TIM, the variables has a time index t. In the following, we present the other models, in their principal variables, the solution are represented by the order of jobs executed in the schedule.

5.3 Relative-Order-Indexed Model1 (ROIM1)

This model uses binaries $y^{\theta}_{j^{'},j}$ and $z^{\theta}_{j^{'},j}$ as decision variables. $y^{\theta}_{j^{'},j} = 1$ if $j^{'}$ is executed immediately before j on θ; $z^{\theta}_{j^{'},j} = 1$ if $j^{'}$ is executed before j on θ. The relative order of job $j^{'}$ and j is implied in $y^{\theta}_{j^{'},j}$ and $z^{\theta}_{j^{'},j}$ if they are on the same machine.

Different formulations of this model can be seen in Błażewicz et al. (1991) and Unlu and Mason (2010) for parallel scheduling problems without precedence constraints. We introduce S_j and C_j for precedence constraints by adding (5.2b), and we propose the following formulation:

$$y^{\theta}_{j^{'},j} \leq z^{\theta}_{j^{'},j}, \quad \forall j^{'}, j \in \mathbb{J}^T, \theta \in \mathbb{M}^T \tag{5.2a}$$

$$M(1 - z^{\theta}_{j^{'},j}) + S_j \geq C_{j'}, \quad \forall j^{'}, j \in \mathbb{J}^T, \theta \in \mathbb{M}^T \tag{5.2b}$$

$$M(1 - z^{\theta}_{j^{'},j}) \geq \sum_{\theta' \neq k} \sum_{q} (z^{\theta'}_{j^{'},j} + z^{\theta'}_{q,j} + z^{\theta'}_{q,j^{'}} + z^{\theta'}_{j,q} + z^{\theta'}_{j^{'},q}), \quad \forall j^{'}, j \in \mathbb{J}^T, \theta \in \mathbb{M}^T \tag{5.2c}$$

$$\sum_{\theta} (y^{\theta}_{j^{'},j} + y^{\theta}_{j,j^{'}}) \leq 1, \quad \forall j^{'}, j \in \mathbb{J}^T \tag{5.2d}$$

$$\sum_{\theta} \sum_{j^{'}} y^{\theta}_{j^{'},j} = 1, \quad \forall j \in \mathbb{J}^T \tag{5.2e}$$

$$\sum_{\theta} \sum_{j} y^{\theta}_{j^{'},j} = 1, \quad \forall j^{'} \in \mathbb{J}^T \tag{5.2f}$$

(5.2a) ensures that $z^{\theta}_{j^{'},j} = 1$ if $y^{\theta}_{j^{'},j} = 1$. (5.2c), (5.2d), (5.2e) and (5.2f) enforce each job to have exactly one predecessor and one successor. Notice that in (5.2e) and (5.2f), each job has to be executed before (after) some other job. In practice, some dummy jobs are created to represent jobs after (before) the last (first) executed jobs on each machine.

In fact, (5.2e) and (5.2f) convey the same meaning: if every job follows another (except the first one), then every job has a follower (except the last one). If we remove one of (5.2e) and (5.2f), experimental results show that the model still works well, but slower. So, we kept them. We call them *redundant constraints*.

5.3.1 Relative-Order-Indexed Model 2 (ROIM2)

We present another relative-order-indexed model, which uses binaries $z^{\theta}_{j^{'},j}$ and $x^{\theta}_{j^{'}}$ as decision variables. $x^{\theta}_{j} = 1$ if j is on θ.

To associate these two variables, Low et al. (2006) and Gao et al. (2006) use the following non-linear constraint for job shop problems:

$$x^{\theta}_{j^{'}} x^{\theta}_{j} = z^{\theta}_{j^{'},j} + z^{\theta}_{j,j^{'}}, \quad \forall j^{'}, j, \theta$$

The version for job shop problem in Özgüven et al. (2010) is linear. However, it introduces new integer variables. Here, we proposed a new formulation, which is linear and does not introduce new integer variables:

$$\sum_{\theta} x_j^{\theta} = 1, \quad \forall j \in \mathbb{J}^T \tag{5.3a}$$

$$C_{j'} \leq S_j + M(1 - z_{j',j}^{\theta}), \quad \forall j^{'}, j \in \mathbb{J}^T, \theta \in \mathbb{M}^T \tag{5.3b}$$

$$M x_{j'}^{\theta} \geq \sum_{j} (z_{j',j}^{\theta} + z_{j,j'}^{\theta}), \quad \forall j^{'} \in \mathbb{J}^T, \theta \in \mathbb{M}^T \tag{5.3c}$$

$$z_{j',j}^{\theta} + z_{j,j'}^{\theta} \leq 1, \quad \forall j, j^{'} \in \mathbb{J}^T, \theta \in \mathbb{M}^T \tag{5.3d}$$

$$M(z_{j',j}^{\theta} + z_{j,j'}^{\theta}) \geq x_{j'}^{\theta} + x_j^{\theta} - 1, \quad \forall j^{'}, j \in \mathbb{J}^T, \theta \in \mathbb{M}^T \tag{5.3e}$$

(5.3a) forces each job to be executed once. (5.3b) ensures that $C_{j'} \leq S_j$ if $z_{j',j} = 1$. (5.3c) considers one case: $x_{j'}^{\theta} = 0$ then $\forall j \in \mathbb{J}^T$, $z_{j',j}^{\theta} = 0$. (5.3c) means that only one of $z_{j',j}^{\theta}$ and $z_{j,j'}^{\theta}$ takes value 1. (5.3e) works when both $j^{'}, j$ are on θ, and forces one to precede the other. We remind the reader that M is a relative large number.

5.3.2 Absolute-Order-Indexed Model (AOIM)

We consider another model which uses $\beta_{\theta,j}^{l}$ as its principal variable. The variable equals 1 if j is the l^{th} job on θ, where l is the absolute-order.

It was originally designed for parallel scheduling problem without precedence constraints by Błażewicz et al. (1991). To add precedence constraints for job shop problems, Demir and İşleyen (2013) introduces T_{θ}^{l} to the model, which is the starting time of the l^{th} job of θ, and x_j^{θ} to indicate the machine of jobs.

Here, we propose a similar formulation, where we removed the variables x_j^{θ} from the formulation of (Demir and İşleyen, 2013):

$$T_{\theta}^{l+1} - T_{\theta}^{l} \geq p_j \beta_{\theta,j}^{l}, \quad \forall l \leq n, j \in \mathbb{J}^T, \theta \in \mathbb{M}^T \tag{5.4a}$$

$$T_{\theta}^{l} + M(1 - \beta_{\theta,j}^{l}) \geq S_j, \quad \forall l \leq n, j \in \mathbb{J}^T, \theta \in \mathbb{M}^T \tag{5.4b}$$

$$T_{\theta}^{l} \leq M(1 - \beta_{\theta,j}^{l}) + S_j, \quad \forall l \leq n, j \in \mathbb{J}^T, \theta \in \mathbb{M}^T \tag{5.4c}$$

$$\sum_{j} \beta_{\theta,j}^{l} \leq 1, \quad \forall l \leq n, \theta \in \mathbb{M}^T \tag{5.4d}$$

$$\sum_{\theta} \sum_{l} \beta_{\theta,j}^{l} = 1, \quad \forall j \in \mathbb{J}^T \tag{5.4e}$$

(5.4e) forces each job to be executed once. (5.4d) ensures that only one job can be executed as the l^{th} job on θ. (5.4c), (5.4b) and (5.4a) works when $\beta_{\theta,j}^{l} = 1$, they guarantee $T_{\theta}^{l} = S_j$ and $T_{\theta}^{l+1} - T_{\theta}^{l} = p_j$.

5.3.3 Compact Model (CM)

We propose a new order-indexed model here. It can be seen as an improved ROIM. It uses $\delta_{j',j}$, which equals 1 if $j^{'}$ is executed before j on the same machine, and x_j^{θ} as decision variables.

$$C_{j'} - S_j \leq M(1 - \delta_{j',j}), \quad \forall j^{'}, j \in \mathbb{J}^T \tag{5.5a}$$

$$M(2 - x_{j'}^{\theta} - x_j^{\theta}) + \delta_{j',j} + \delta_{j,j'} \geq 1, \quad \forall j^{'}, j \in \mathbb{J}^T, \theta \in \mathbb{M}^T \tag{5.5b}$$

$$M(2 - x_{j'}^{\theta_1} - x_j^{\theta_2}) \geq \delta_{j',j} + \delta_{j,j'}, \quad \forall j^{'}, j \in \mathbb{J}^T, \theta_1, \theta_2 \in \mathbb{M}^T \text{ and } \theta_1 \neq \theta_2 \tag{5.5c}$$

$$\sum_{\theta} x_j^{\theta} = 1, \quad \forall j \in \mathbb{J}^T \tag{5.5d}$$

(5.5a) connects $\delta_{j',j}$, $C_{j'}$ and S_j (if $\delta_{j',j} = 1$ then $C_{j'} \leq S_j$). When both $j^{'}, j$ are on θ, (5.5b) forces one precedes the other. (5.5d) makes each job be executed exactly once. (5.5c) sets $\delta_{j',j}$ and $\delta_{j,j'}$ as 0 when $j^{'}, j$ are on different machines.

Notice that when minimizing $C_{\max}$, (5.5c) is redundant because when $j^{'}, j$ are on different machines, $\delta_{j',j} = 1$ or $\delta_{j,j'} = 1$ can not reduce $C_{\max}$. However, it helps to give $\delta_{j',j}$ a comprehensible meaning ($\delta_{j',j} = 1$ if and only if $j^{'}$ precedes j on the same machine) and has a positive impact on the model.

5.4 Test Result and Analysis

We tested the models with benchmarks we built from precedence constraints in (Kolisch and Sprecher, 1996). The instances are originally designed for scheduling problems with shared resources and multiple resource consumption, including renewable, non-renewable or doubly constrained resources. Jobs have informations such as due date, release date and tardiness cost etc. We removed all the constraints other than precedences constraints, set the number of machines differently, and set a unique requirement of resources consumption.

The platform we used is: IBM ILOG CPLEX Optimization Studio V12.6.0 on Intel Core i7-4600U @2.10GHz. We compared their average time consumed to solve instances with different scales of jobs and machines in Table 5.1, where TC means average time consumed to solve the instances; '-' means the model did not solve any instance on this scale within 6000sec.

m	4			3	5	10
n	15	30	60	30	30	30
CM	0.92	1.22	13.32	0.98	0.34	0.06
ROIM2	1.77	6.41	97.23	6.32	4.88	1.66
AOIM	3.19	14.66	-	14.32	13.98	14.53
ROIM1	15.41	79.23	-	9.36	4.28	3.32
TIM	48.39	-	-	-	40.26	20.94

Table 5.1: Performance of various models

Model	BV	IV	C
CM	n^2+nm	$2n$	$n^2(m^2+m+1)+n$
ROIM2	n^2m+nm	$2n$	$2n^2m+2mn+n$
AOIM	n^2m+nm	$2n+nm$	$3n^2m+mn+n$
ROIM1	$2n^2m$	$2n$	$3n^2m+n^2+2n$
TIM	$nm\sum p_j$	$2n$	$n^2m\sum p_j+mn\sum p_j+mn+n$

Table 5.2: Number of variables and constraints, where BV/IV/C means number of binary variables/integer variables/constraints

We find that when m is set as 4, the models took longest time for solving. As it can be seen, CM stays ahead of the others and requires less space. The different speed of models results mostly from the different decision variables used.

The Time-Indexed Model (TIM) requires an estimation of an upper bound of $C_{\max}$. It can be as large as $\sum p_j$ for the worst case (single machine) in practice, which leads to large amounts of variables. In our test, we set $t_{\max}$ as the exact value of $C_{\max}$, which minimizes the number of variables. In fact, the number of variables could be extremely large if p_j is not an integer. However, TIM is still the slowest even for instances of unit-processing-time jobs. Both ROIM2 and ROIM2 use $O(n^2m)$ binary variables: ROIM1 uses two 3-dimension variables, while ROIM2 and AOIM use only one. AOIM is the only one which requires extra integer variable. The fastest model CM uses only 2-dimension binary variables and requires fewest variables, which may be the principal advantage of CM.

CM does not have the fewest constraints. The fewest constraints are not necessarily another advance. There exists some redundant constraints, such as (5.2f) and (5.5c), which exert positive effects on models, see Table 5.3.

However, the redundant constraints does not certainly play a positive role in all models. For example, we find that adding $T_\theta^l = \sum_{l'=1}^{l-1}\sum_j \beta_{\theta,j}^{l'} p_j$ to AOIM, which fix the value of T_θ^l immediately when the decision variable $\beta_{\theta,j}^l$ is fixed, makes the model dramatically slower, see Table 5.4.

In addition to redundant constraints, the variables C_j which could totally be replaced by $S_j + p_j$, can be seen as redundant variables. We tested and find that they improve the models.

	ROIM1		CM	
	with	without	with	without
$n = 15$	15.41	17.89	0.92	1.10
$n = 30$	79.23	90.91	1.22	1.41

Table 5.3: Performance of ROIM1 and CM with/without redundant constraints, average time in sec.

	with	without
$n = 15$	10.01	3.19
$n = 30$	46.92	14.66

Table 5.4: Performance of AOIM with/without redundant constraints, average time in sec.

5.5 Conclusion and Perspective

We adapted models to the parallel scheduling problem with precedence constraints and proposed a new one which outperforms the others in running test.

TIM, ORIM1, ORIM2 and AOIM have an original version for flexible job-shop problem (FJSP). The only difference is that FJSP has dedicated machine constraints, which means the machines are not identical. It's not hard to add these constraints to our new model and adapt it to FJSP. We conjecture that it can still win out over the others. For parallel scheduling problem, they can also be easily adapted to other objective function such as the total completion timeor $L_{\max}$ by adding a constraint on C_j. The model can be employed not only for the scheduling problems but for other ordered assignment problems as well.

Besides, the time to solve does not depend merely on instance's scale, but also on number of machines, the processing time of jobs, and the shape of precedence constraints. The model is helpful to study experimentally how the different precedence constraints impact the time to solve.

Model	with	without
CM	1.22	1.79
ROIM2	6.41	7.88
AOIM	14.66	19.01
ROIM1	79.32	95.17

Table 5.5: Models' performance solving instances of $n = 30, m = 4$ with/without C_j

Part IV

Conclusion

In this thesis, we discussed the complexity of parallel scheduling problems with precedence constraints and the way to solve it.

We first put forward a proof of $\mathcal{NP}$-completeness for $P|p_j = 1, \text{in-tree}|\sum C_j$. We also proved that scheduling unit-processing-time jobs under level-order or opposing-forests precedence constraints is NP-complete, for both $C_{\max}$ and $\sum C_j$ whether it is preemptive or not. Then, we adapt the proof to other scheduling problems, such as out-tree and opposing forests. This work updates the earlier study of scheduling problems' complexity in literature.

On the other hand, we developed an algorithm for the $\mathcal{NP}$-complete scheduling problem $P|p_j = p, i.t.|\sum C_j$, with a complexity $O(h^m)$, where h is the height of the in-tree. This algorithm is both theoretically and experimentally faster than Baptiste's algorithm (Baptiste et al., 2004). Afterwards, we consider solving the general scheduling problem $P|prec|C_{\max}$ by linear integer programming models. We adapted some existing models which are originally designed for other problems, and propose a new one, which uses less variables and runs faster on LP solver such as Cplex.

All this work refines the complexity limits and closes open cases. This could be particularly useful to classify a huge set of problems that derivate from parallel scheduling, such as hybrid shop problem. We also proposed resolution improvement to help decision tools being faster and more efficient.

Add to those extensions and applications, several theoretical research avenues also exist. In terms of complexity, there exist still open problems, such as $P|p_j = 1, i.t., pmtn|\sum C_j$, that we have to study. We pointed out that our proof cannot be immediately adapted to this preemptive version by proposing a counter-example. So, this need further investigation. In addition, when the number of machines is fixed, some problems are still open such as $Pm|p_j = p, ch., r_j|\sum C_j$. Another open problem, $Pm|p_j = p, l.o.|\sum C_j$, is conjectured to be polynomial solvable.

Regarding resolution methods and practical aspects to have efficient methods for real problems, we proposed some research avenues on exponential algorithms, which could be extended to other areas, such as shop scheduling problems and other sequencing problems. For example, in ride-sharing systems such as Uber, Didi, Lyft etc, demand list what a driver picks can be modeled as a sequence of jobs that a machine executes.

On another aspect, scheduling problems are not the only ones which depends on the shape of the graph. We may consider other OR problems where a graph is involved, such as network restoration problem (Averbakh, 2017), in which the shortest recovery time to reach every nodes has to be found. In such a field, the complexity may also vary when the graph takes a particular shape (e.g. tree) or some parameters are fixed. So, there clearly exist open problems too.

Bibliography

I. Averbakh. Minimizing the makespan in multiserver network restoration problems. *Networks*, 70(1):60–68, 2017. 48

J. Błażewicz, M. Dror, and J. Węglarz. Mathematical programming formulations for machine scheduling: A survey. *European Journal of Operational Research*, 51(3):283–300, 1991. 43, 44

P. Baptiste and V. G. Timkovsky. On preemption redundancy in scheduling unit processing time jobs on two parallel machines. *Operations Research Letters*, 28(5):205–212, 2001. 8

P. Baptiste, P. Brucker, S. Knust, and V. G. Timkovsky. Ten notes on equal-processing-time scheduling. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 2(2):111–127, 2004. 8, 30, 48

P. Brucker. *Scheduling algorithms*, volume 3. Springer, 2007. 4

P. Brucker, M. R. Garey, and D. S. Johnson. Scheduling equal-length tasks under treelike precedence constraints to minimize maximum lateness. *Mathematics of Operations Research*, 2(3):275–284, 1977. 8, 9, 27

P. Brucker, J. L. Hurink, and S. Knust. A polynomial algorithm for $P|p_j = 1, r_j, outtree|\sum C_j$. *Math. Methods Oper. Res.*, 2001. 8, 9, 27

P. Brucker, S. Heitmann, and J. Hurink. How useful are preemptive schedules? *Operations Research Letters*, 31(2):129–136, 2003. 8, 23

J. Bruno, E. G. Coffman Jr, and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17(7):382–387, 1974. 7

M. Chardon and A. Moukrim. The coffman–graham algorithm optimally solves uet task systems with over-interval orders. *SIAM Journal on Discrete Mathematics*, 19(1):109–121, 2005. 9

E. G. Coffman and R. L. Graham. Optimal scheduling for two-processor systems. *Acta informatica*, 1(3): 200–213, 1972. 7

R. W. Conway, W. L. Maxwell, and L. W. Miller. Theory of scheduling. eading. *Massachussets: Addison-Wesley*, 1967. 2

Y. Demir and S. K. İşleyen. Evaluation of mathematical models for flexible job-shop scheduling problems. *Applied Mathematical Modelling*, 37(3):977–988, 2013. 44

K. Djellab. Scheduling preemptive jobs with precedence constraints on parallel machines. *European journal of operational research*, 117(2):355–367, 1999. 8

D. Dolev and M. K. Warmuth. Scheduling precedence graphs of bounded height. *Journal of Algorithms*, 5 (1):48–59, 1984. 8

D. Dolev and M. K. Warmuth. Profile scheduling of opposing forests and level orders. *SIAM Journal on Algebraic Discrete Methods*, 6(4):665–687, 1985. 4, 8

J. Du, J. Y. Leung, and G. H. Young. Scheduling chain-structured tasks to minimize makespan and mean flow time. *Information and Computation*, 92(2):219–236, 1991. 7, 8

P. C. Fishburn. *Interval orders and interval graphs: A study of partially ordered sets*. John Wiley & Sons, 1985. 4

J. Gao, M. Gen, and L. Sun. Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm. *Journal of Intelligent Manufacturing*, 17(4):493–507, 2006. 43

M. R. Garey and D. S. Johnson. Computers and intractability: a guide to np-completeness, 1979. 11

M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002. 11

M. R. Garey, D. S. Johnson, E. Tarjan, and M. Yannakakis. Scheduling opposing forests. *SIAM Journal on Algebraic Discrete Methods*, 4(1):72–93, 1983. 8, 9, 27

T. F. Gonzalez and D. B. Johnson. A new algorithm for preemptive scheduling of trees. *Journal of the ACM (JACM)*, 27(2):287–312, 1980. 9, 27

R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979. 4

T. C. Hu. Parallel sequencing and assembly line problems. *Operations research*, 9(6):841–848, 1961. 8, 9, 27, 30

Y. Huo and J. Y.-T. Leung. Minimizing mean flow time for UET tasks. *ACM Transactions on Algorithms (TALG)*, 2(2):244–262, 2006. 8

S. Knust and P. Brucker. Complexity results for scheduling problems, 2009. http://www2.informatik.uni-osnabrueck.de/knust/class/. 8

R. Kolisch and A. Sprecher. Psplib – a project scheduling problem library. *EUROPEAN JOURNAL OF OPERATIONAL RESEARCH*, 1996. URL `http://www.om-db.wi.tum.de/psplib/format.html`. 42, 45

W. Kubiak, D. Rebaine, and C. Potts. Optimality of hlf for scheduling divide-and-conquer uet task graphs on identical parallel processors. *Discrete Optimization*, 6(1):79–91, 2009. 9

E. L. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. In *Annals of Discrete Mathematics*, volume 2, pages 75–90. Elsevier, 1978. 7

E. L. Lawler. Preemptive scheduling of. precedence-constrained jobs on parallel machines. In *Deterministic and stochastic scheduling*, pages 101–123. Springer, 1982. 8, 9, 27

J. K. Lenstra and A. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35, 1978. 8

J. K. Lenstra, A. R. Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of discrete mathematics*, 1:343–362, 1977. 7, 42

C. Low, Y. Yip, and T.-H. Wu. Modelling and heuristics of fms scheduling with multiple objectives. *Computers & operations research*, 33(3):674–694, 2006. 43

R. H. Möhring. Computationally tractable classes of ordered sets. In *Algorithms and order*, pages 105–193. Springer, 1989. 8

A. Moukrim. Non-preemptive profile scheduling and quasi interval orders. *Electronic Notes in Discrete Mathematics*, 3:133–139, 1999. 8

C. Özgüven, L. Özbakır, and Y. Yavuz. Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling*, 34(6):1539–1548, 2010. 43

C. H. Papadimitriou and M. Yannakakis. Scheduling interval-ordered tasks. *SIAM Journal on Computing*, 8 (3):405–409, 1979. 8

M. L. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2012. v, 2, 5, 7, 15

D. Prot and O. Bellenguez-Morineau. A survey on how the structure of precedence constraints may change the complexity class of scheduling problems. *Journal of Scheduling*, pages 1–14, 2017. 2, 8, 9

C. S. Thomalla. Job shop scheduling with alternative process plans. *International Journal of Production Economics*, 74(1):125–134, 2001. 42

J. D. Ullman. NP-complete scheduling problems. *Journal of Computer and System sciences*, 10(3):384–393, 1975. 2, 7

J. D. Ullman. Complexity of sequencing problems. *Computer and Job-Shop Scheduling Theory, EG Co man, Jr.(ed.)*, 1976. 7

Y. Unlu and S. J. Mason. Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. *Computers & Industrial Engineering*, 58(4):785–800, 2010. 43

G. Weiss and M. Pinedo. Scheduling: Theory, algorithms, and systems, 2012. 6