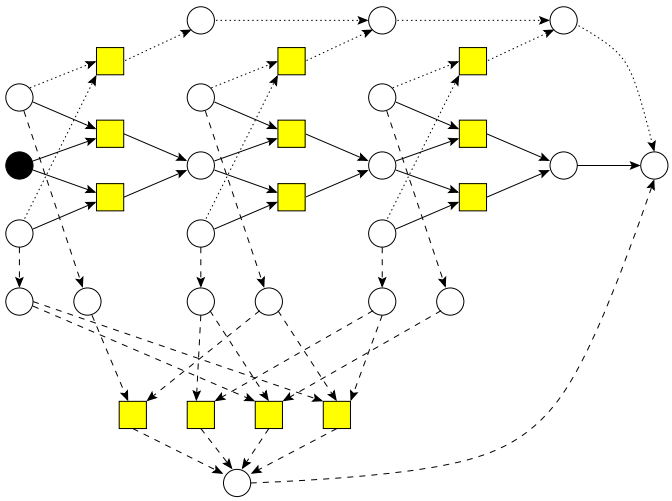
Scheduling with AND/OR-Networks



Vanessa Kääb

Berlin 2003
D83

Scheduling with AND/OR-Networks

vorgelegt von
Dipl.-Math. Vanessa Kääb
aus Konstanz

Vom Institut für Mathematik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Berichter: Prof. Dr. Rolf H. Möhring,
Technische Universität Berlin
Berichter: Prof. Dr. Thomas Erlebach,
Eidgenössische Technische Hochschule Zürich

Tag der wissenschaftlichen Aussprache: 29. April 2003

gefördert durch die DFG im Rahmen des
Europäischen Graduiertenkollegs “Combinatorics, Geometry, and Computation”
(GRK 588/2)

Berlin 2003
D83

ACKNOWLEDGEMENTS

When I finished my diploma in Mathematics at the University of Konstanz in 2000, Dorothea Wagner, the supervisor of my diploma thesis, called my attention to a research position in the European Graduate Program "Combinatorics, Geometry, and Computation". At her advice and encouragement, I successfully applied for this position within the group of Rolf Möhring. This thesis is the outcome of three years of research and the contributions of a number of people.

First of all, I am grateful to my supervisor Rolf Möhring for his support, his guidance, and his valuable comments. Rolf Möhring inspired my work in numerous fruitful discussions and I greatly benefited from his expertise in combinatorial optimisation and scheduling in particular. Part of this thesis is based on joint work with Rolf Möhring and Martin Skutella and I wish to thank Martin Skutella in particular for helpful advice and "the right idea" for solving several problems.

My special thanks go to Thomas Erlebach, who does not only take the second assessment of this thesis but in fact took an active part in its development. During the second year of my doctoral studies I spent half a year as an academic guest visiting the research group of Thomas Erlebach at the ETH Zurich. In addition to his tutorial during this time he affectively assisted writing up this thesis and establishing part of the results presented in Chapter 3.

As a member of the European Graduate Program "Combinatorics, Geometry, and Computation", I profited from the surpassing possibilities for professional as well as social exchange, starting with the weekly lectures, diverse schools and courses, to the annual international workshop of the programme. The graduate programme is supported by the German National Science Foundation (DFG) (grant GRK 588/2) and I thank everybody who made the programme possible.

I wish to thank Thomas Erlebach, Volker Kaibel, Martin Skutella, Frederik Stork, Andreas Fest, Christian Liebchen, Heiko Schilling, and Carsten Lange for their careful proof-reading of different parts of the manuscript. I am indebted to Marc E. Pfetsch for providing the pictures displayed in Figure 1 and to Christoph Eyrich who became an indispensable adviser for all questions concerning LaTeX.

Moreover I am thankful to the colleagues in the work groups of Rolf Möhring and Günter Ziegler for creating such a unique and stimulating research atmosphere, by providing a centre of expertise on the one hand and a great social community on the other hand.

Berlin, March 2003 *Vanessa Kääb*

CONTENTS

Introduction **1**

1 AND/OR Constrained Scheduling **9**
1.1 Preliminaries 9
1.1.1 Basic Notation 10
1.1.2 The Three Field Classification Scheme 16
1.2 AND/OR-Networks 17
1.2.1 Feasibility 17
1.2.2 The Earliest Start Schedule 19
1.2.3 Directed Hypergraphs and Related Work 23

2 Criticality in AND/OR-Networks **27**
2.1 Critical Jobs in Project Networks 27
2.2 Critical Jobs in AND/OR-Networks 28
2.3 Structural Properties of Critical Jobs and Sets 32
2.3.1 Clutters and Blocking Clutters 32
2.3.2 Systems of Critical Sets 33
2.3.3 The Max-Flow-Min-Cut Property of Critical Sets 36
2.3.4 Invariants of Critical Sets 42
2.4 The Complexity of Finding Critical Jobs and Sets 57
2.4.1 The Complexity of Constructing Critical Sets 57
2.4.2 The Complexity of Identifying Critical Jobs 59
2.4.3 Criticality and Monotone Boolean Functions 65
2.5 Criticality in Specially Structured AND/OR-Networks 71
2.6 A Quantifying Analysis of Critical Sets 79
2.6.1 Changing the Makespan 79
2.6.2 The Linear Time-Cost Trade-off Problem 81

3 Scheduling with AND/OR Precedence and Machine Constraints **87**
3.1 List Scheduling 88
3.2 The Makespan 90
3.2.1 Scheduling on Identical Parallel Machines 91
3.3 The Total Weighted Completion Time 95
3.3.1 Scheduling on One Machine 95

3.3.2 Scheduling on Identical Parallel Machines 110

4 Local Search **113**
4.1 Introduction to Local Search 113
4.2 A Canonical Neighbourhood for AND/OR-Networks 117
4.3 Properties of the Solution Space 118
4.4 Local Search and Longest Paths 121

Bibliography **127**

Symbol Index **135**

Index **139**

Zusammenfassung **143**

Curriculum Vitae **145**

INTRODUCTION

For several decades now, scheduling has been an attractive research area within combinatorial optimisation. In general, scheduling is the process of assigning a timetable or plan of production to a set of tasks, commonly called *jobs*. Depending on the respective application, a variety of models have been developed in scheduling theory.

Figure 1: Images of the Potsdamer Platz in Berlin: The left image shows the construction site of the Potsdamer Platz in 1998, when it was Europe's biggest construction site, while the right image shows the Potsdamer Platz now, in 2003.

One of those standard models is captured by *project scheduling*. As the nomenclature already suggests, this model describes whatever one could call a project. A project might be a construction, a chemical process, a spacecraft launch or a transportation problem of heavy load. The left picture of Figure 1 shows an example of such a project: the construction site of the Potsdamer Platz in Berlin in 1998. This construction site was the largest construction site in Europe at that time. The right picture shows the result of the project, the Potsdamer Platz now, four years later. It should be clear that there is much to optimise and many activities to schedule in such a gigantic project.

The different activities that have to be undertaken in a project are represented by the jobs. We assume that we know in advance how long every activity in the project will take to be finished. This time requirement is represented by *processing times* on the jobs which are usually nonnegative. In most problems there arise interdependencies between certain jobs, that is some jobs can only be started after

some other jobs have been finished. On a construction site one can think of plastering the walls, which can certainly only be done after the walls have been built up. These dependencies are modeled by *precedence constraints*, commonly depicted in a *precedence graph*. In addition, jobs compete for common limited resources, such as commodities, machines, or labour, commonly abstracted as *resource constraints*. Probably the most relevant goal in project scheduling is to complete the project as early as possible. In scheduling theory the goal of the problem under consideration is called the *objective* or the *objective function* as in general, this goal can be expressed as a function of the completion times of the jobs. Besides the project completion time, the so-called *makespan*, the objective might be to meet one or several deadlines, or to minimise the sum of the completion times of the jobs.

In *machine scheduling* the focus is on the major restriction of the scheduling problem, the machine constraints. A set of jobs has to be scheduled on one or several machines in order to optimise the given objective function. The machines represent processors, workers or assembly lines. As such, the main applications of machine scheduling can be found in computer science and industrial production. Of course the classification between project and machine scheduling problems is floating. Although most parts of this thesis are in the field of project scheduling, Chapter 3 explicitly focuses on machine scheduling problems.

In this thesis we consider a special model of scheduling, namely scheduling with AND/OR-networks. Basically, an AND/OR-network is a directed graph or digraph for short. In contrast to standard digraphs, an AND/OR-network has two different sorts of nodes, AND- and OR-nodes, which represent the following constraints. An AND-node is just a normal node with the meaning that the job it represents can be processed as soon as every job preceding it in the graph has been completed. An OR-node on the other hand can be scheduled as soon as at least one of its predecessors has been finished. These generalised precedence constraints provide a good model for a variety of applications. We want to discuss two of them in more detail to motivate our effort on the subject.

The first example is *resource constrained project scheduling*. As already mentioned above, in this application the task is to schedule a set of jobs such that all precedence constraints between the jobs are respected. In addition, the jobs have individual demands for certain limited resources. The precedence constraints are given by the precedence graph. The *resource constraints* can be given as a set of resources, the amount that is available of each resource and the demand of each job for each of the resources. In this setting, we have to take care that at any point in time, the total demand for each of the resources of the jobs that are processed at that time does not exceed the availability of the resource. Again for an example, consider a construction site. If we have to paint the walls of five rooms in a flat, but we have only employed two painters, we cannot do it in par-

allel. These resource restrictions can also be represented in a more abstract way by *minimal forbidden sets*. A set of jobs without any precedence dependencies between them is called *forbidden*, if their total demand of some resource exceeds the available amount of that resource. Thus painting the five rooms, for example, is a forbidden set of jobs. A forbidden set is *minimal forbidden*, if any proper subset of it can be scheduled in parallel. In our case, painting any three of the five rooms constitute a minimal forbidden set, as our two painters can do any two rooms at the same time. A straightforward approach to resolve the resource conflicts given by a set of minimal forbidden sets would be to simply fix for every forbidden set a precedence relation between two of the jobs in the set. This technique reduces the resource constrained scheduling problem to a precedence-only constrained scheduling problem, which is easy to solve. Nevertheless, we might make a bad choice by fixing these relations. A less strict way to resolve the resource conflict, is to select one job in each minimal forbidden set, which then has to wait for the completion of at least one of the other jobs in its set. We call the job that is selected the *waiting job*. The pair consisting of a forbidden set without its waiting job and the waiting job is called a *waiting condition*. Of course we can still make a bad choice with the waiting job, but at least it does not have to wait for one previously selected job in the forbidden set, but can instead start as soon as an arbitrary one of the other jobs in the forbidden set has been completed. The precedence graph together with a set of waiting conditions can be represented by an AND/OR-network. For every waiting condition we introduce an OR-node which is a direct successor of each job in the forbidden set except the waiting job. The waiting job is itself a direct successor of the OR-node. In this model, the precedence constraints represented by the AND/OR-network capture the constraints of the waiting conditions and thus resolve the resource conflicts.

The second application are assembly and disassembly problems. As the name already suggests, those problems arise in manufacturing processes, for example. Consider the problem of disassembling a certain product on an assembly line. Due to the geometry of the product, there might be different possibilities how to actually disassemble the product or parts of it. A little example is given in Figure 2. The left figure represents the product and we want to remove part number 7, to repair it, for example. There exist four possible directions to remove a part of the product, indicated by the thick black arrows. The right figure presents the AND/OR-network that corresponds to the precedence constraints appearing in our problem. The circles are AND-nodes and the shaded squares are the OR-nodes of the network. To remove part 7, we can either remove part 5 and 6 to the south or part 4 to the west or part 3 to either the north or the east. To remove part three to the north, we need to remove part 1 to the north, and to remove part 3 to the east we have to remove part 2 to the east first. A schematic overview over different variants of assembly problems has been presented in Goldwasser and Motwani

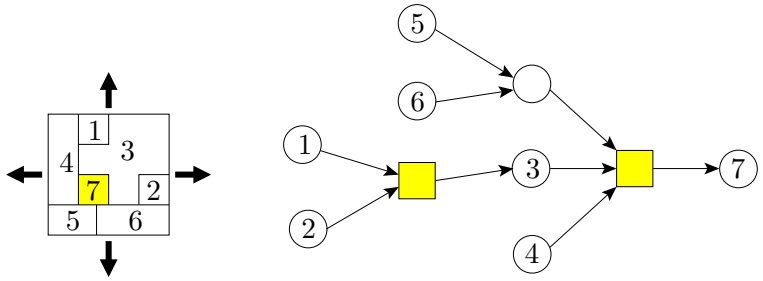


Figure 2: The left figure displays a product that has to be disassembled. The thick arrows mark the possible directions in which a part of the product may be removed. The right figure presents the AND/OR-network that corresponds to the precedence relations occurring in this disassembly instance.

(1999).

Chapters 3 and 4 are motivated by the problem of scheduling jobs restricted by AND/OR precedence constraints such that either the makespan or the sum of the weighted completion times of the jobs are minimised. The focus of Chapter 2 in contrast is slightly different. Consider again the construction site of the Potsdamer Platz displayed in Figure 1. At an early stage of the project, the project plan and the scheduling model representing this plan have to be designed. At this stage, the exact data is not known and has to be estimated. Even after thorough considerations, it is not possible to take everything into account. Especially in those long term undertakings, there is a high uncertainty in the processing times of the jobs, the availability of resources, and even the actual interdependencies between the jobs. This problem is on one hand attended to by *stochastic scheduling*. In stochastic scheduling, (parts of) the input data is assumed to be subject to random fluctuations. Thus the solution is not a schedule but a plan of procedure, a so-called *scheduling policy*, based on the past evolution of the instance. We will not pursue this approach and refer to the respective literature for this matter. Another possibility to attend to uncertainties in the input data is what could be called a *sensitivity analysis* of the network. We restrict our focus on changes in processing times of jobs. Some of the jobs in a project are not *critical* in the sense that, if they take longer, they do not increase the makespan. Whereas other jobs may directly influence the project completion time. Most of the time any delays in the project completion involve high costs so that it might be better to invest some money into the prolonged critical job, to keep its processing time fixed. In any case, a thorough analysis of the problem structure can provide the project planner with a useful tool to handle any external influences. Those questions have been discussed intensively for the case of standard precedences. Whereas for AND/OR-networks, to our knowledge, only the so-called *time-cost trade-off*

problem has been addressed, which can be attributed to this area of research. We will focus on a sensitivity analysis in AND/OR-networks in Chapter 2.

Outline of the Thesis

This thesis is divided into four chapters that will be described in detail in the following. The first chapter is supposed to introduce the reader into the basic concepts that will be used in all other chapters. Chapters 2 to 4 are essentially independent of each other. Throughout the thesis we consider AND/OR precedence constrained jobs. First we examine structural issues of AND/OR-networks and the behaviour of schedules in AND/OR-networks subject to changes in processing times. The next chapter will tighten the restrictions imposed among the jobs by additionally taking machine constraints into account. Chapter 4 then goes into the other direction and considers a relaxation of the AND/OR constraints, in which we are allowed to exchange the waiting job.

Chapter 1 is dedicated to give an introduction into scheduling with AND/OR-networks. First we specify the data of the scheduling model used and review basic graph theoretic notation needed to describe our model. In Remark 1.1.2 important assumptions about the structure of the AND/OR-networks considered in this thesis are listed and justified. A standard classification scheme will enable us to describe efficiently the respective model under consideration. Section 1.2 concentrates on AND/OR-networks and previous work on scheduling in AND/OR-networks, including the presentation of an algorithm to check for *feasibility* of AND/OR-networks and a Dijkstra-like algorithm to compute an *earliest start schedule* for the jobs. Important features of the earliest start schedule are that it is optimal for several objectives such as the makespan and that it is monotone as a function on the processing times. We close the chapter by a short presentation of *directed hypergraphs* and publications on directed hypergraphs related to our work. A directed hypergraph is a graph theoretic concept equivalent to an AND/OR-network.

Chapter 2 presents a thorough qualitative and quantitative analysis of AND/OR-networks, respectively their earliest start schedule with respect to a strictly positive processing time vector. We start with a review of the concept of criticality in standard precedence graphs. Inspired by the different notions of criticality in standard networks, we define four different types of critical jobs and sets in AND/OR-networks, namely *cut-*, *path-*, *delay-*, and *bulk-critical* jobs and sets. A cut-critical set is an inclusion-minimal set of jobs with the property that an arbitrarily small decrease of the processing times of all jobs in the set results in a

decrease of the makespan. A path-critical set in contrast is inclusion-minimal with the property that it preserves the makespan, even if the processing times of all jobs not in the path-critical set are decreased. Delay- and bulk-critical sets are defined similarly but with respect to an increase in processing times. A job is critical, if it is included in a critical set. We prove that in each of the four cases, the system of all critical sets of one type is a clutter. In particular, the cut-critical sets are the blocking clutter of the path-critical sets and the delay-critical sets are the blocking clutter of the bulk-critical sets. Thus, a cut-critical job is also path-critical and a delay-critical job is also bulk-critical, but in general, for a job being cut-critical does not imply being delay-critical and being delay-critical does not imply being cut-critical. Both pairs of blocking clutters do not have the *Max-Flow-Min-Cut property* and we prove that the gap between a maximum flow on the path-critical sets and the minimal costs of a cut-critical set, for example, is at least as big as the integrality gap of SET COVER. This fact already indicates that the time-cost trade-off problem, which we will treat later, might be hard. Next we extend the notion of criticality to OR-nodes as well by defining the *closure* of path- and bulk-critical sets. For every critical set we prove that the change in processing times does not affect the start times of the jobs in the set. This property is the main ingredient for obtaining a number of later results. A purely structural representation of path- and bulk-critical sets closes the basic studies of critical sets and jobs.

The next section is dominated by complexity matters. We notice that it is easy to find a critical set in each of the four cases. In contrast to this, it is NP-complete to decide whether a certain job is cut-, path-, delay-, or bulk-critical. This result is established by a reduction from the satisfiability problem (SAT), where it is remarkable that two different reductions are needed for the four cases, one for cut- and path-critical jobs and one for delay- and bulk-critical jobs. For all other results about critical jobs or sets, the proofs for the four cases can be accomplished in a way similar to each other. In an excursion into logic, we ascertain a close relation between critical sets in AND/OR-networks and minimal fulfilling assignments for monotone Boolean functions. We establish a complexity result for minimal fulfilling assignments of monotone Boolean functions which is new to our knowledge: It is NP-complete to decide whether a variable is true in some inclusion-minimal truth assignment.

In Section 2.5 we briefly discuss structurally restricted AND/OR-networks. In AND/OR-networks forming an *in-tree*, the four notions of criticality are equivalent and it is easy to decide whether a job is critical. For *N-free* AND/OR-networks it turns out that a path-critical (and thus also cut-critical) job is bulk-critical (and delay-critical), but the other implication is not true in general. Considering the AND/OR-network as a hypergraph, we establish a slightly different notion of N-freeness. By giving appropriate counter-examples we show that if an AND/OR-network is N-free in the hypergraph notation, again neither of the implications

hold.

We conclude the chapter by a quantitative analysis of AND/OR-networks with respect to changes in processing times of critical sets. Cut- and delay-critical sets have a direct unbiased impact on the makespan. If the processing times of the jobs in a cut-critical set are decreased by some small amount, the makespan decreases by exactly the same amount. The equivalent holds for an increase in the processing times of the jobs in a delay-critical set. Another issue that belongs to the quantitative analysis of AND/OR-networks is the *time-cost trade-off* problem. We introduce the linear time-cost trade-off problem and discuss the properties of the time-cost trade-off curve on a significant example. The time-cost trade-off curve is piecewise linear and nonincreasing, but neither monotone nor continuous, like in the case of standard precedence constraints. In addition, the DEADLINE PROBLEM is NP-complete, which was proved by Stork (2001).

Chapter 3 discusses the problems of minimising the makespan and the total weighted completion time of a set of AND/OR precedence constrained jobs on one or several identical parallel machines. First we present Graham's well known *list scheduling* algorithm, which will be the major ingredient of the subsequent approaches to the stated problems. We consider the two objective functions in one section each, again divided into the one machine and the several machine case. In each section we review known results for the case of no and standard precedence constraints among the jobs and then present known and new results for AND/OR-precedence constraints.

The makespan objective is trivial to solve on one machine if the jobs are not restricted at all, but also if standard or AND/OR precedence constraints are imposed among the jobs. The problem on identical parallel machines is NP-complete even without precedence constraints and just two machines. Nevertheless, Graham's list scheduling provides a 2-approximation for scheduling precedence constrained jobs. Gillies and Liu (1995) present a 2-approximation for scheduling jobs on an acyclic AND/OR-network. This algorithm first transforms the AND/OR-network into a standard precedence graph and then applies list scheduling. We extend the algorithm of Gillies and Liu to general feasible AND/OR-networks.

Minimising the total weighted completion time on one machine is less trivial for precedence constrained jobs than the makespan. It is well known that the basic problem without any additional constraints can be solved optimally by *Smith's rule*. The total weighted completion time of standard precedence constrained jobs has been approximated with guarantee 2 with various techniques, which resist any application to AND/OR precedence constraints. For minimising the total completion time, that is the special case of unit weights, we prove that list scheduling with *shortest processing time rule* is a $\sqrt{n}$-approximation, where n is the number

of jobs. This bound is tight. The case with arbitrary weights seems to be even harder. By a reduction of LABEL COVER proposed by Goldwasser and Motwani (1999) together with an improved inapproximability result for LABEL COVER by Dinur and Safra (1999) we show that minimising the total weighted completion time to within a factor of $2^{\log^{1-1/\log\log^c n} n} n$ of the optimum is NP-hard for any $c < 1/2$. We prove that list scheduling with shortest processing time rule is a simple n-approximation for the problem and that this bound is again tight.

Chapter 4 considers a relaxation of the AND/OR precedence constraints, where it is allowed to exchange the direct successor of an OR-node against one of its direct predecessors. The idea is to use neighbourhood search to find an AND/OR-network that minimises the makespan. We first give a short introduction into local search and then propose a canonical neighbourhood on AND/OR-networks. The main result in this chapter is that the feasible networks are connected under the proposed neighbourhood. We briefly introduce the job shop problem, where local search has been applied quite successfully. The basic neighbourhoods used for the job shop problem resemble our canonical neighbourhood for AND/OR-networks. We close the chapter and the thesis by a short discussion about possible strategies of neighbourhood search in AND/OR-networks.

The thesis is essentially self-contained and we will introduce most of the notation needed. Nevertheless we expect the reader to be familiar with the basic concepts of graph theory, combinatorial optimisation, complexity theory, logic, and local search. For an introduction to graph theory we refer to Bollobás (1990), Jungnickel (1991), Diestel (2000), and West (2001). A thorough overview over combinatorial optimisation is presented in Schrijver (2003), but we also refer the reader to Korte and Vygen (2000), Nemhauser and Wolsey (1988), and Trotter (1992). The textbooks of Garey and Johnson (1979), Ausiello, Crescenzi, Gambosi, Kann, Marchetti-Spaccamela, and Protasi (1999), and Hochbaum (1997) cover our needs for complexity theory. For an introduction to logic and Boolean functions, we refer to Wegener (1987) and Prestel (1992). An introduction into and an overview of application of local search in combinatorial optimisation is presented in Aarts and Lenstra (1997).

CHAPTER 1

AND/OR CONSTRAINED SCHEDULING

This chapter is supposed to acquaint the reader with the basic notations used throughout this thesis and to present some already known results. The first section introduces fundamental definitions and describes a widely used classification scheme for scheduling problems. Thereafter, we concentrate on AND/OR-networks, paying attention to standard matters such as feasibility, structural properties and scheduling with AND/OR-networks. Most of the definitions and algorithms presented in the section are taken from Möhring, Skutella, and Stork (2000b). Parts of this paper have been published as an extended abstract in Möhring, Skutella, and Stork (2000a). We assume the reader to be familiar with the very basic concepts of graph and order theory. For an introduction to graph theory and the concepts used in the thesis we refer to the textbooks of West (2001), Diestel (2000), or Bollobás (1990). For order-theoretic matters, an introductory course into partially ordered sets can be found in Trotter (1992), for example. An order-theoretic view on scheduling is presented in Möhring and Radermacher (1989). In Möhring (1989) the focus is laid on special classes of ordered sets which allow efficient solutions for problems that are NP-hard in the general case. In later chapters we will make also use of some of these special classes and illustrate to which extent they are able to facilitate the problems under consideration. Most of the graph and order-theoretic notation introduced in Section 1.2 is taken from Möhring (1989).

1.1 Preliminaries

In this section we want to introduce the reader to the basics of scheduling theory that will be used throughout the thesis. We use standard notations that can be found in the related literature. For a compact introduction to scheduling, we refer to Brucker (1998) where the basic notation as well as the classification scheme can be found. In addition, classical examples of scheduling problems and their solutions are discussed and a simple survey of computational complexity is presented.

1.1.1 Basic Notation

A scheduling problem in this thesis is given by a set of data which will be specified in the following. Most of the time, we will consider problems which are given by a subset of the information but for the sake of completeness, we will now introduce all the terms used. A scheduling instance consists of a set of n *jobs* $V = \{1, 2, \ldots, n\}$. With each job j we associate

- a (strictly) positive *processing time* p_j,
- a *release date* $r_j \geq 0$,
- and a positive *weight* ω_j.

By $p_{\min} := \min\{p_j \mid j \in V\}$ we denote the *minimum processing time* of a job in V. In addition to the jobs, a number m of identical parallel *machines* will be given. The machines all run with the same speed and can process any job. Of course, at any point in time, each machine can process only one job and every job can only be processed by one machine. We will only consider the non-preemptive case. Once a job is started on a certain machine, it will be finished on that machine without any interruption. In contrast to that, in the preemptive case a job may be interrupted and its processing is resumed later on any machine.

The jobs are subject to two different classes of constraints. On one hand, standard *precedence constraints* are given. Such a precedence relation usually means that one job j has to wait for the completion of another job i since, for example, the execution of j depends on parts of the output of i. Those precedence relations can be represented by a directed acyclic graph $G = (V(G), E(G))$, the so-called *precedence digraph*, where each vertex in $V(G)$ corresponds to a job. For simplicity, we will not explicitly distinguish between a job and the vertex representing it. An edge $(i, j) \in E(G)$ represents the constraint that j has to wait for the completion of i. In a feasible realisation of such a problem, all jobs have to be executed in accordance to the partial order induced by G. Each job $j \in V$ has to wait for the completion of all its predecessors in the partial order and thus will also be called an *AND-node*.

On the other hand, we also allow for precedence relations of the form that a job j has to wait for the completion of only one predecessor. Those restrictions cannot be captured by the classical precedence constraints described above. The model of standard precedence constraints can be generalised by the introduction of a set W of waiting conditions. A *waiting condition* $w \in W$ is an ordered pair $w = (X, j)$, where $X \subseteq V$ is a set of jobs and $j \in V \setminus X$ is the *waiting job*. The waiting job j can be processed as soon as one of the jobs in X has been completed. A generalised version of the waiting conditions is to allow constraints

of the form that job j may start only after at least k jobs out of a pre-given set $X \subseteq V \setminus \{j\}$ of (at least k) jobs are completed. These generalised precedence constraints provide a smooth transition between standard precedence constraints, represented by $k = |X|$, and waiting conditions where $k = 1$. We will restrict our focus to the two extreme cases although the general waiting conditions cannot be modeled in this setting proceeding on the assumption that the processing times of the jobs are strictly positive.

The standard precedence constraints together with the waiting conditions can be represented by an *AND/OR-network* in the following way: for every waiting condition $w = (X, j) \in W$, we introduce an *OR-node*, which we will denote by w again. For every $x \in X$, we introduce a directed edge (x, w). In addition, there is a directed edge (w, j) for the waiting job j. To model the required constraints correctly, we impose the rule that an OR-node w can be scheduled as soon as any of its predecessors $x \in X$ is completed. An OR-node $w \in W$ can be considered as a pure dummy node with processing time $p_w = 0$, release date $r_w = 0$, and weight $\omega_w = 0$. The precedence digraph G together with the OR-nodes can be depicted in one AND/OR-network $N = (V \cup W, E)$, where V is the set of jobs and W the set of waiting conditions. The set of edges contains the edges $E(G)$ and the additional edges incident to the OR-nodes.

One motivation for scheduling with AND/OR-networks originates from resource constrained project scheduling, where a set of jobs has to be scheduled subject to precedence and resource constraints. The precedence constraints are represented by the precedence digraph as described. One possibility to represent the resource constraints is by a set $\mathcal{F}$ of *minimal forbidden sets*. A *forbidden set* $F \in \mathcal{F}$ is a set of jobs without any precedence relations among them but that cannot be scheduled in parallel as their consumption of some resource exceeds the availability of that resource. A forbidden set is *minimal*, if every proper subset can be scheduled in parallel. For an extended introduction into the representation of resource constraints in project scheduling we refer to Stork and Uetz (2000). For our considerations it suffices to know that we can represent resource constraints by minimal forbidden sets. For every minimal forbidden set $F \in \mathcal{F}$, we select one job $j \in F$ and introduce a waiting condition of the form $(F \setminus \{j\}, j)$. This resolves the resource conflicts, as $F \setminus \{j\}$ can be scheduled in parallel and as soon as one job in $F \setminus \{j\}$ is completed, also j can be processed.

Another application of scheduling with AND/OR-networks can be found in assembly problems. In Goldwasser and Motwani (1997), the authors consider the problem of partially disassembling a product to reach a single part or component. A part might be accessed only if certain other parts have been removed before. Starting from another geometric direction, the same part might also be reached, if some other parts are removed before. Those restrictions can be modeled by AND/OR precedence constraints. Most of the research in this thesis is

motivated by resource constrained project scheduling. However, in Chapter 3 assembly problems are the major application.

For AND/OR-networks most of the standard graph theoretical notation can be used. We will briefly recall the required notation. In addition we will make some simplifying, but not restricting, assumptions about the structure of an AND/OR-network.

Given an AND/OR-network $N = (V \cup W, E)$, then for an edge $(u, v) \in E$, we call u a *direct predecessor* of v and, correspondingly, v *direct successor* of u. The set of direct or *immediate* predecessors of a vertex v is denoted by $ImPred(v) := \{u \in V \cup W \mid (u, v) \in E\}$, the set of direct or immediate successors is denoted by $ImSucc(v) := \{u \in V \cup W \mid (v, u) \in E\}$. A *directed path* P_{uv} from a vertex u to a vertex v, or u-v-path for short, is a sequence of nodes $u = u_0, u_1, \ldots, u_k = v$, where $k = 0$ is possible, such that $(u_{i-1}, u_i) \in E$ for all $i = 1, \ldots, k$. The *cardinality* of a u-v-path is k, that is the number of edges (u_{i-1}, u_i) on the path. The sum of processing times, $\sum_{i=0}^{k} p_{u_i}$, of the jobs on a path P_{uv} is called the *length* of P_{uv}. If $u = v$ and $k > 0$, the path is called a *cycle* and if $u_i \neq u_j$ for all $i \neq j$, the path is *simple* or *acyclic*. An AND/OR-network is called *simple* or *acyclic* if it does not contain a cycle. For a given vertex v, $Succ(v) := \{u \in V \cup W \mid \text{there exists a path from } v \text{ to } u\}$ denotes the set of all *successors* of v, while $Pred(v) := \{u \in V \cup W \mid \text{there exists a path from } u \text{ to } v\}$ denotes the set of all *predecessors* of v in N. As we allow $k = 0$, every vertex is a predecessor and successor of itself and thus $v \in Succ(v) \neq \emptyset$ and $v \in Pred(v) \neq \emptyset$ for all $v \in V \cup W$. There are several possibilities to define the size of an AND/OR-network N. Our definition is motivated by the resource constrained project scheduling application, where the input is the precedence digraph $G = (V(G), E(G))$ and a set of minimal forbidden sets, $\mathcal{F} = \{F_1, \ldots, F_f\}$. The size of this input is $|V(G)| + |E(G)| + \sum_{i=1}^{f} |F_i|$. According to this approach, we define the *size* of $N = (V \cup W, E)$ as $size(N) = |N| := |V| + |E|$. Note that $|E| = |E(G)| + \sum_{i=1}^{f} |F_i|$ and $|V| = |V(G)|$. Throughout the thesis we will consider directed graphs, directed AND/OR-networks, and directed hypergraphs, which will be introduced later in Section 1.2.3, and we will omit explicitly writing "directed" in most cases.

It will be useful to have a unique start vertex and a unique end vertex in the network. Therefore, we will introduce a common start vertex s which is a direct predecessor of every vertex j with $ImPred(j) = \emptyset$ and a common end vertex t that is a direct successor of every other vertex $j \in V$. We assume $p_s = p_t = 0$, $r_s = r_t = 0$, and $\omega_s = \omega_t = 0$. We add s and t to the vertex set, $V = \{s = 0, 1, \ldots, n, n + 1 = t\}$. Node s will also be called the *source* and t the *sink* of network N. To avoid unnecessarily expensive notation, we will not explicitly exclude s and t, when for example working with the strictly positive processing

times of the jobs in V. It should be clear from the context, when they are included in an argument and when not. They are pure dummy nodes to ease the explanation and should not bloat up notation.

Consider the AND/OR-network presented in the left picture of Figure 1.1. To illustrate that s and t are artificial dummy nodes, they are drawn with dotted lines, as well as all edges incident to them. Although t is a direct successor of all jobs, we will not display any transitive edges incident to t, which has the advantage that it emphasises the jobs in the AND/OR-network that have no job as a direct successor. In the thesis we will draw OR-nodes as shaded squares, whereas jobs are displayed as circles.

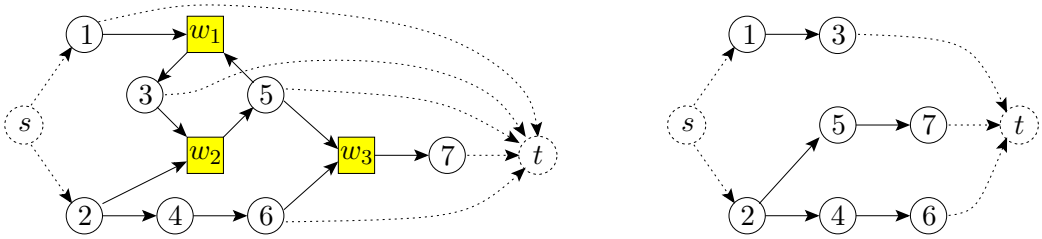


Figure 1.1: An AND/OR-network N and a realisation of N.

After we have identified the problem data, we can define the actual task of a scheduling problem.

Definition 1.1.1. *For a given scheduling instance, a* schedule *is a nonnegative vector of starting times* $S = (0 = S_s, S_1, S_2, \ldots, S_n, S_t)$. *We call a schedule* feasible, *if it respects all of the required constraints, that is*

1. $S_v \geq r_v$ *for all* $v \in V \cup W$ *(*release constraints*)*
2. $S_j \geq \max\{S_v + p_v \mid (v,j) \in E\}$ *for all* $j \in V$ *(*AND-constraints*)*
3. $S_w \geq \min\{S_v + p_v \mid (v,w) \in E\}$ *for all* $w \in W$ *(*OR-constraints*)*
4. $|\{j \in V \mid S_j \leq t < S_j + p_j\}| \leq m$ *at any time* $t \geq 0$ *(*machine constraints*)*

The value or cost of a schedule in general is expressed by some function in the completion times of the jobs. Therefore, let $C = (C_1, C_2, \ldots, C_n)$ be the *vector of completion times* associated with the schedule S, where $C_j = S_j + p_j$ for $j = 1, \ldots, n$. The so called *objective function* measures the value of a proposed schedule. There are a variety of naturally occurring objective functions, such as the minimal project duration, generally called the makespan in scheduling theory. In this thesis we will focus on minimising one of the following three objective functions,

- the *makespan* $C_{\max} = \max\{C_j \mid j \in V\}$,
- the *total completion time* $\sum_{j \in V} C_j$, and
- the *total weighted completion time* $\sum_{j \in V} \omega_j C_j$.

By construction we get that $C_{\max} = S_t = C_t$. The total completion time and the average completion time, $\frac{1}{n} \sum_{j \in V} C_j$, are equivalent objectives as well as the total weighted completion time and the average weighted completion time, $\frac{1}{n} \sum_{j \in V} \omega_j C_j$.

We can make several simplifying and standardising assumptions about the structure of an AND/OR-network N. Most of them follow naturally from the description of the AND/OR-network we have presented so far. Nevertheless, AND/OR-networks stemming from other applications might have different structures. We want to restrict the structure of a network and explain, how this structure can be achieved by simple modifications, if it is not fulfilled by the AND/OR-network under consideration.

Remark 1.1.2. *For an AND/OR-network $N = (V \cup W, E)$ we may assume the following structural properties without loss of generality.*

1. *Every OR-node $w \in W$ has processing time $p_w = 0$.*
2. *Every OR-node $w \in W$ has exactly one direct successor.*
3. *The direct successor j of an OR-node $w \in W$ is a job, thus an AND-node.*
4. *For every OR-node $w = (X, j) \in W$ there are no precedence relations with respect to $G = (V, E(G))$ between any two jobs incident to w, that is the AND-nodes in $X \cup \{j\}$.*

First note that an AND/OR-network N stemming from a precedence digraph G and a set of waiting conditions W automatically fulfills points 1 to 3. Nevertheless, we can transform any AND/OR-network such that it fulfills points 1 to 3. Suppose some OR-node $w \in W$ has a strictly positive processing time $p_w = r > 0$. We introduce an AND-node i with edge (w, i) and replace the edges (w, j) by edges (i, j) for all $j \in ImSucc(w)$. We set $p_i = r$ and $p_w = 0$. If the network contains an OR-node $w \in W$ with $k > 1$ direct successors $j_1, \ldots, j_k$, we introduce k copies of w: $w_1, \ldots, w_k$. For each $i = 1, \ldots, k$, copy w_i is assigned the direct successor j_i of w as its direct successor. On the other hand, if the direct successor u of an OR-node w is itself an OR-node, we can remove w and make all the direct predecessors of w additional direct predecessors of u.

Point 4 is motivated by AND/OR-networks derived from instances of resource constrained project scheduling with minimal forbidden sets, where this property

is inherited from the structure of the forbidden sets. Suppose network $N = (V \cup W, E)$ does not have the required property and let $w = (X, j)$ be an OR-node with a precedence relation between two nodes in $X \cup \{j\}$. We have to distinguish three cases, depicted in Figure 1.2.

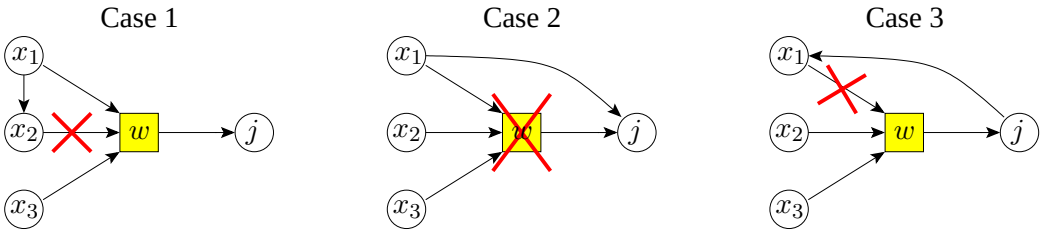


Figure 1.2: The three possibilities for precedence relations between jobs incident to an OR-node w and how to settle the problem.

First, assume that there exists a relation between two jobs in X and without loss of generality let $(x_1, x_2) \in E(G)$. We then can remove the edge (x_2, w). For the start time of x_2 we know that $S_{x_2} = \max\{S_v + p_v \mid (v, x_2) \in E\} \geq S_{x_1} + p_{x_1}$. Due to the strictly positive processing times of all jobs, it follows that $S_w = \min\{S_{x_i} + p_{x_i} \mid x_i \in X\} = \min\{S_{x_i} + p_{x_i} \mid x_i \in X \setminus \{x_2\}\}$. The described situation is presented in Figure 1.2, Case 1. Now assume that one of the jobs in X is a direct predecessor of j, say $(x_1, j) \in E(G)$. This corresponds to Case 2 in the figure above. By an easy calculation it follows that we can remove w itself and all its incident edges. For the OR-node w it holds that $S_w = \min\{S_{x_i} + p_{x_i} \mid x_i \in X\} \leq S_{x_1} + p_{x_1}$. For the start time of j we therefore get that $S_j = \max\{S_v + p_v \mid (v, j) \in E\} \geq S_{x_1} + p_{x_1} \geq S_w + p_w$ and thus w cannot influence the start time of its direct successor j. The last case to consider is Case 3 of Figure 1.2. Suppose that one of the jobs in X, say x_1, is a direct successor of j. If there exists some $x_i \in X \setminus \{x_1\}$, we can remove the edge (x_1, w) as $S_{x_1} \geq \max\{S_v + p_v \mid (v, x_1) \in E\} \geq S_j + p_j > S_j \geq \min\{S_{x_i} + p_{x_i} \mid x_i \in X\}$. Thus the minimum completion time of the predecessors of w cannot be attained by x_1 and the edge (x_1, w) is redundant. If $X \setminus \{x_1\} = \emptyset$, we get the contradiction that $S_j \geq S_w = S_{x_1} + p_{x_1} > S_{x_1} \geq S_j + p_j > S_j$ and thus there does not exist a feasible schedule. In any of the three possible cases, the argumentation holds as well if the relation is a transitive relation in G. Then the considered edge, (x_1, x_2) in Case 1 for example, is replaced by a directed nonempty path of jobs. All needed inequalities obviously also hold in this situation.

We remark that these operations can only entail a polynomial increase of the size of the network N and can be performed in polynomial time. In Möhring, Skutella, and Stork (2000b), the authors make the additional assumption that the AND/OR-network is bipartite with respect to the two classes of vertices V and W.

This can be achieved by simply introducing an OR-node for every edge between two jobs in V. Although this assumption proves to be useful for certain considerations we will not apply this here. We need the distinction between a standard edge stemming from the precedence digraph G and an edge incident to an OR-node later.

To specify efficiently the scheduling problem under consideration, we will give a short introduction into a classification scheme in the next section.

1.1.2 The Three Field Classification Scheme

Due to the large variety of scheduling problems, it is convenient to classify them schematically like Graham, Lawler, Lenstra, and Rinnooy Kan (1979) or Lawler, Lenstra, Rinnooy Kan, and Shmoys (1993). We will not describe the full scheme here, as we only use a very restricted number of parameters. However, as it is a very nice, clear and short way of describing the problem under consideration, we will give a quick introduction into the $\alpha|\beta|\gamma$-notation of Lawler, Lenstra, Rinnooy Kan, and Shmoys (1993). The three fields describe the following data adapted to AND/OR-networks, where we have restricted the specification to the values occurring in this thesis.

- α specifies the machine environment, where α can take one of the following terms with the specified meaning.
  - $\circ$: no machine constraints, there are as many machines as needed
  - 1: one machine
  - Pm: m identical parallel machines, where m is a constant
  - P: identical parallel machines, their number is part of the input
- β describes the job characteristics and can take none or several of the following expressions, separated by a comma.
  - r_j: the jobs have release dates
  - $p_j = 1$: all jobs have unit processing time
  - $prec$: standard (AND) precedence constraints
  - $ao\text{-}prec$: AND/OR precedence constraints
- γ specifies the objective function which is, as mentioned above, one of the following.
  - $C_{\max}$: the makespan

- $\sum C_j$: the total completion time
- $\sum \omega_j C_j$: the total weighted completion time

In the general framework, for each of the fields there are a number of other specifications. In the machine case, one can also distinguish between $\alpha = Q$ or $\alpha = R$ which means that there are uniform parallel or unrelated parallel machines. The job characteristics can be extended to $pmtn$, d_j, or $tree$, which denominates the preemptive case, given due dates for the jobs, or tree-like precedence relations, respectively. For the objective function, there are also other specifications, such as the *maximum lateness* $L_{\max} = \max\{C_j - d_j \mid j \in V\}$ or the *total tardiness* $\sum_{j \in V} \max\{0, C_j - d_j\}$, where d_j are due dates given in the problem instance.

For example, the problem of scheduling jobs with unit processing times and precedence constraints on one machine with the objective to minimise the total completion time would be specified as $1|prec, p_j = 1| \sum C_j$. One of the problems we will focus on in this thesis is minimising the makespan of a set of AND/OR precedence constrained jobs, which can be expressed by $\circ|ao\text{-}prec|C_{\max}$.

1.2 AND/OR-Networks

In the last section we have specified the problem data and its notation. Due to the three field classification scheme presented, we are able to denote the problem instance under consideration briefly and easy to recognise. In this section we concentrate on AND/OR-networks and scheduling with AND/OR precedence constraints.

1.2.1 Feasibility

In scheduling with standard precedence constraints and strictly positive processing times, a necessary and sufficient condition for the existence of a schedule is that the precedence digraph is acyclic. In the case of AND/OR-networks, cycles in general do not interfere with feasibility. Here, the non-appearance of a *generalised cycle* is a necessary and sufficient condition for the existence of a valid schedule. We need some more notation.

An AND/OR-network $N' = (V' \cup W', E')$ is a *sub-network* of $N = (V \cup W, E)$, if $V' \subseteq V$, $W' \subseteq W$, $E' \subseteq E$, and for every edge $(u, v) \in E'$ it holds that $u, v \in V' \cup W'$. For an edge set $E' \subseteq E$ the AND/OR-network $N' = N(E') = (V' \cup W', E')$ is the *(edge) induced* sub-network of N if $V' \cup W' = \bigcup_{(u,v) \in E'} \{u, v\}$. For a vertex set $V' \cup W' \subseteq V \cup W$, $N' = N(V' \cup W') = (V' \cup W', E')$ is called *(vertex) induced* , if $E' = \{(u, v) \in E \mid u, v \in V' \cup W'\}$. In an AND/OR-network $N = (V \cup W, E)$ a *generalised cycle* is a nonempty

sub-network $N' = (V' \cup W', E')$ of N such that

- $ImPred(j) \cap (V' \cup W') \neq \emptyset$ for each $j \in V'$ and
- $ImPred(w) \subseteq V'$ for each $w \in W'$.

Definition 1.2.1. *An AND/OR-network $N = (V \cup W, E)$ is called* feasible *if it does not contain a generalised cycle as sub-network.*

In Figure 1.3 an AND/OR-network is presented that contains a generalised cycle. The nodes enclosed in the gray square, the jobs $3, 4$ and 5, together with the three waiting conditions, form a generalised cycle corresponding to the definition above. Obviously, for any vector of strictly positive processing times, it is not possible to find a schedule for the jobs without violating any of the precedence constraints. This should illustrate the difference between a standard cycle, which does not necessarily collide with feasibility, as for example contained in the network presented in Figure 1.1 and a generalised cycle as depicted in Figure 1.3.

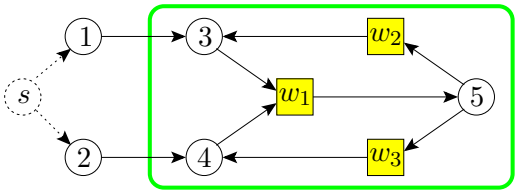


Figure 1.3: An AND/OR-network containing a generalised cycle.

Remember that the AND/OR-network $N = (V \cup W, E)$ contains the precedence digraph G, representing the standard precedence constraints, as a subgraph. We will consider extensions of the partial order described by G that "respect" the OR-constraints. A *realisation* of an AND/OR-network $N = (V \cup W, E)$ is a partial order $R = (V, <_R)$ such that

- $i <_R j$ for each $(i, j) \in E$ with $i, j \in V$ and
- for each $w = (X, j) \in W$, there exists $i \in X$ with $i <_R j$.

A linear or total order $L = (V, <_L)$, which is a realisation of N, is called *linear realisation*[1]. It is easy to see that the existence of a (linear) realisation also guarantees the feasibility of an AND/OR-network. As a summary of several results presented in Möhring, Skutella, and Stork (2000b), we can state the following theorem. For an example examine Figure 1.1, where an AND/OR-network N is presented in the left figure and a possible realisation of N in the right figure. A linear realisation of N would be $L_1 = 1, 2, 3, 4, 5, 6, 7$, but also $L_2 = 1, 3, 5, 7, 2, 4, 6$.

[1]For the layout realisation, Goossens, Mittelbach, and Samarin (1994) is a good companion

Algorithm 1: Linear Realisation

Input: AND/OR-network $N = (V \cup W, E)$

Output: A list L of jobs in V

```
L := ∅;
while there exists job j ∈ V with ImPred(j) = ∅ do
    add j at the end of L;
    foreach OR-node w ∈ ImSucc(j) ∩ W do
        remove w from the network N;
    remove j from the network N;
return L;
```

Theorem 1.2.2 (Möhring, Skutella, and Stork (2000b)). *For any AND/OR-network $N = (V \cup W, E)$ the following statements are equivalent.*

1. *N is feasible, that is it does not contain a generalised cycle.*
2. *N has a realisation R.*
3. *N has a linear realisation L.*

For the sake of completeness, we present a modified version of Algorithm 1 in Möhring et al. (2000b). It constructs a linear list L in a simple greedy fashion.

Theorem 2.3.1 in Möhring et al. (2000b) states that an AND/OR-network N is feasible if and only if the list L, constructed by Algorithm 1, contains all jobs of V. The proof is quite straightforward and will be omitted. It is easy to see that the algorithm can be implemented in time linear in the size of N. Thus, Algorithm 1 provides us with a quick feasibility check for AND/OR-networks, generating a linear realisation if the AND/OR-network is feasible and a generalised cycle if it is infeasible as a certificate.

1.2.2 The Earliest Start Schedule

In the last section, we have seen that checking feasibility of an AND/OR-network is easy. The feasibility of an AND/OR-network is also referred to as the *structural* feasibility of the given problem instance. On the other hand, we are interested in a feasible schedule S. A schedule is feasible if it respects all given constraints. It might exist, even if the AND/OR-network is not feasible. This can happen, if there are cycles of zero or negative length in the network. As we have restricted our considerations to strictly positive processing times on the jobs, this case cannot occur in our model. Thus, we do not have to distinguish

between structural feasibility of the AND/OR-network and the existence of a feasible schedule. We will consider the very simple setting of $\circ|ao\text{-}prec|C_{\max}$ and investigate how to compute a schedule for it.

Obviously, $S = (\infty, \ldots, \infty)$ is a feasible schedule as it fulfills all the inequalities stated for a schedule in Section 1.1.1. It is also quite easy to observe that if $S = (S_1, \ldots, S_n)$ and $S' = (S'_1, \ldots, S'_n)$ are feasible schedules, then also the componentwise minimal schedule $S'' = (\min\{S_1, S'_1\}, \ldots, \min\{S_n, S'_n\})$ is a feasible schedule. It follows that there exists a (unique) componentwise minimal schedule, the so-called *earliest start schedule* ES. We assume strictly positive job processing times, thus an earliest start schedule can be computed by a modification of Dijkstra's shortest path algorithm. In the literature several such algorithms have been proposed, among others by Knuth (1977), Adelson-Velsky and Levner (1999), and Möhring, Skutella, and Stork (2000b). Algorithm 2 computes an earliest start time schedule for a feasible AND/OR-network with strictly positive processing times on the jobs. It is a slightly modified version of the earliest start algorithm presented in Möhring, Skutella, and Stork (2000b). Möhring et al. (2000b) assume without loss of generality the AND/OR-network N to be bipartite, with the AND-nodes forming one of the vertex classes and the OR-nodes the other. We do not assume this in our setting and therefore we have to change the algorithm slightly.

It is apparent that Algorithm 2 computes the earliest start schedule for a given instance $(N = (V \cup W, E), p)$ and that it can be implemented to run in time polynomial in the size of the AND/OR-network. We refer to Möhring, Skutella, and Stork (2000b), where the correctness and the exact runtime is proven for the presented algorithm.

By definition, the earliest start schedule ES has the following property. For every job $j \in V$, the inequality $S_j \geq \max\{S_v + p_v \mid (v, j) \in E\}$ is fulfilled with equality for at least one edge $(v, j) \in E$ as otherwise j could start earlier. For the same reason, for each OR-node $w \in W$, the inequality $S_w \geq \min\{S_v + p_v \mid (v, w) \in E\}$ is also fulfilled with equality for at least one edge. We call those edges (v, j) and (v, w) for which equality holds *tight*, and we will refer to the nodes v as the *tight immediate predecessors* of j and w, $ImPred^{tight}(j)$ and $ImPred^{tight}(w)$, respectively. Accordingly, a predecessor $v \in Pred(u)$ of a node u is called a *tight predecessor* of u, if there exists a tight path from v to u, that is a path containing only tight edges. The set of tight predecessors of a vertex u is denoted by $Pred^{tight}(u)$. Tight successors and tight immediate successors are defined in the obvious way. The set of tight edges will be denoted by $E^{tight} \subseteq E$ and the sub-network induced by the tight edges will be denoted by $N^{tight} = N(E^{tight})$. In addition we have to take care about the difference between the start and the completion times of the nodes incident to nontight edges in E.

Algorithm 2: Earliest Start Schedule

Input: feasible AND/OR-network with strictly positive processing times p on the jobs in V

Output: earliest start schedule $ES = (S_1, \ldots, S_n)$

foreach *OR-node* $w \in W$ **do**
  set $S_w := \infty$;

foreach *job* $j \in V$ **do**
  if $ImPred(j) = \emptyset$ **then**
    set $S_j := 0$;
    foreach *unmarked OR-node* $w \in ImSucc(j) \cap W$ **do**
      set $S_w := \min\{S_w, S_j + p_j\}$;
    foreach *job* $k \in ImSucc(j) \cap V$ **do**
      `ProcessJob(`k`)`;
  else set $S_j := \infty$;

while *there exists an unmarked OR-node* $w \in W$ **do**
  choose an unmarked OR-node w with minimum start time S_w;
  mark w and `ProcessJob(`$ImSucc(w)$`)`;

return $(S_1, \ldots, S_n)$;

Procedure: `ProcessJob(`j`)`
reduce indegree of j by 1;
if *indegree*$(j) = 0$ **then**
  set $S_j := \max\{S_i + p_i \mid i \in ImPred(j)\}$;
  foreach *unmarked OR-node* $w \in ImSucc(j) \cap W$ **do**
    set $S_w := \min\{S_w, S_j + p_j\}$;
  foreach *job* $k \in ImSucc(j) \cap V$ **do**
    `ProcessJob(`k`)`;

Therefore we define the *slack* of an edge $e \in E$ to be

$$slack(N, p, e) := \begin{cases} S_v - (S_u + p_u) & \text{for } e = (u, v) \in E \text{ and } v \in V \\ S_u + p_u - S_v & \text{for } e = (u, v) \in E \text{ and } v \in W \end{cases}$$

Clearly, $slack(N, p, e) = 0$ for every tight edge $e \in E^{tight}$. The *slack* of a network $(N = (V \cup W, E), p)$ can then be defined as the minimum of the minimum positive slack and the minimum processing time, that is

$$slack(N, p) := \min\{\, \min\{slack(N, p, e) \mid e \in E \setminus E^{tight}\}\,,\, p_{\min}\,\}$$

Note that $slack(N, p) > 0$ as we only consider strictly positive processing times.

In contrast to many other settings, scheduling jobs with AND/OR precedence constraints but without any machine constraints is easy, as Algorithm 2 produces a feasible schedule and runs in polynomial time. The produced schedule, the earliest start schedule ES, is feasible and it is also optimal for several objective functions. As every job is started as early as possible without violating any constraint, it also finishes as early as possible. Thus, we can make the following statement.

Proposition 1.2.3. *For an instance $(N = (V \cup W, E), p)$ of the problem of scheduling jobs subject to AND/OR precedence constraints, the ES is an optimal schedule for all the three objective functions, the makespan, the total completion time, and the total weighted completion time.*

In this simple setting, the earliest start schedule as well as the three objective functions are monotone. As we will need this property later, we want to state it formally.

Lemma 1.2.4. *For the problem of scheduling a set of jobs V subject to AND/OR precedence constraints, the earliest start schedule is a monotone increasing function in the processing times p of the jobs in V.*

Proof. Consider two vectors of processing times p and q and suppose p dominates q, that is $p_i \geq q_i$ for every $i = 1, \ldots, n$. By the definition of the AND- and OR-constraints, any feasible schedule $S(p)$ for p is also feasible for q. The earliest start schedule $ES(p)$ is then a feasible schedule for q and componentwise minimal for vector p. Thus, the earliest start schedule $ES(q)$ for q can only be smaller than or equal to $ES(p)$ in every component, proving the monotonicity. □

Obviously, the vector of completion times is also monotone increasing in the processing times. As all three objective functions are monotone functions in the vector of completion times, we come to the following conclusion.

Corollary 1.2.5. *For the problem of scheduling AND/OR precedence constrained jobs, the makespan, the total completion time, and the total weighted completion time are monotone increasing functions in the processing times p of the jobs in V.*

In summary we can state that there exists an efficient, polynomial time algorithm to optimally solve each of the three considered problems $\circ|ao\text{-}prec|C_{\max}$, $\circ|ao\text{-}prec|\sum C_j$, and $\circ|ao\text{-}prec|\sum \omega_j C_j$. In Chapter 3, we will see that this nice situation changes completely as soon as, in addition to the AND/OR precedence constraints, machine constraints are imposed.

We want to conclude this chapter with a structural treatment of AND/OR-networks within the theory of hypergraphs.

1.2.3 Directed Hypergraphs and Related Work

A directed hypergraph is a generalisation of the concept of directed graphs. Directed graphs are used for one-to-one relations over finite sets, whereas in many areas of computer science a more general framework of functional relations is needed. Directed hypergraphs model many-to-one or in the more general case even many-to-many relations, thus providing a suitable framework for a number of applications. From a structural point of view, AND/OR-networks are equivalent to directed hypergraphs. Various research groups are and have been working on directed hypergraphs and their properties. Most of this work emanates from a graph theoretic background. Some of the results presented in the literature are closely related to our work. We will discuss structural properties of AND/OR-networks in this thesis that have nice graph theoretic interpretations on hypergraphs. It will be convenient to switch to the hypergraph terminology when examining certain properties of AND/OR-networks. Thus we will give a short introduction to hypergraphs and a brief overview over the related work. The following notations are taken from Gallo and Scutellá (1999).

A *directed hypergraph*, or simply hypergraph, is a pair $H = (V, A)$, where $V = \{1, 2, \ldots, n\}$ is the vertex set and A a set of directed hyperedges. A *directed hyperedge*, or again simply a hyperedge, is a pair $a = (T_a, h_a)$, with $T_a \subseteq V$ and $h_a \in V \setminus T_a$. We call T_a the *tail* and h_a the *head* of a. In contrast to Gallo and Scutellá (1999), we will not consider the case of an edge with an empty tail or an empty head. An even more general framework has been introduced in Gallo, Longo, Pallottino, and Nguyen (1993). In their model, also the head of a hyperedge is a subset of V. They call a hyperedge with a one-element head a *B-arc* and a graph just containing B-arcs as hyperedges a *B-graph*. Thus, the B-graphs of Gallo et al. (1993) correspond to the hypergraphs that we will consider in the following.

Obviously, every directed graph is also a directed hypergraph. In an AND/OR-network N, every OR-node can be interpreted as a hyperedge, where the immediate predecessors of the OR-node form the tail and the immediate successor the head of the hyperedge. More formally, for an AND/OR-network $N = (V \cup W, E)$, we define the *hypergraph of* N, $H(N) = (V, A)$, as follows. The vertex set of $H(N)$ is just the set of jobs V. The edge set is composed of two disjoint classes of hyperedges, $A = A^E \cup A^W$, corresponding to edges and OR-nodes in the AND/OR-network, respectively. The class A^E contains a hyperedge $a(e) = (\{i\}, j)$ for every edge $e = (i, j) \in E$ with $i, j \in V$, that is an edge with both end vertices being jobs. In A^W there exists a hyperedge $a(w) = (ImPred(w), ImSucc(w))$ for every OR-node $w \in W$. It will be convenient to switch to the hypergraph view of an AND/OR-network, as we will see in Section 2.3.4, for example.

A common way to draw hypergraphs is depicted in Figure 1.4. The presented hypergraph corresponds to the AND/OR-network of Figure 1.1.

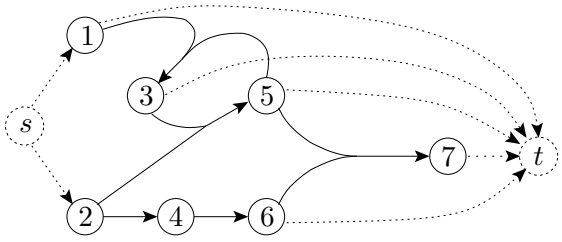


Figure 1.4: The hypergraph of the AND/OR-network of Figure 1.1.

Gallo and Scutellá (1999) define the *size* of a hypergraph H as $size(H) = \sum_{a \in A} |T_a|$. For an AND/OR-network $N = (V \cup W, E)$ and its associated hypergraph we get that $|E| - |W| = \sum_{a \in A} |T_a|$ and thus $size(H(N)) = size(N) - |W| - |V|$. A hypergraph $H' = (V', A')$ with $V' \subseteq V$, $A' \subseteq A$, and for each $(T_a, h_a) \in A'$, $T_a \subseteq V'$ and $h_a \in V'$ is called a *sub-hypergraph* of H. A *path* P_{uv} from u to v in H is a sequence $u = v_0, a_1, v_1, a_2, \ldots, a_k, v_k = v$ with $v_{i-1} \in T_{a_i}$ and $h_{a_i} = v_i$ for all $i = 1, \ldots, k$. If $u = v$ we call the path a *cycle*. A path P_{uv} is *simple* or *acyclic* if it does not contain any cycle as a sub-path, that is $v_i \neq v_j$ for all $i \neq j$, $i, j = 0, \ldots, k$. Obviously, a path in the hypergraph corresponding to an AND/OR-network is the same as a path in the AND/OR-network itself. A hypergraph is called *simple* or *acyclic* if it does not contain a cycle. A *directed hyperpath*, or simply hyperpath, Π_{Rv} from the root set $R \subseteq V$ to a node $v \in V$ in H is a minimal acyclic sub-hypergraph of H containing R and v, such that every node in Π_{Rv}, except the nodes in R, have exactly one incoming hyperedge. We will see that hyperpaths correspond to certain critical sets in AND/OR-networks that will be defined later.

A *cut*, or more precisely an s-t-cut $C_{st} = (V_s, V_t)$ of a hypergraph $H = (V, A)$ is a partition of the vertex set V into two disjoint subsets V_s and V_t such that $s \in V_s$ and $t \in V_t$. For a given cut C_{st}, the *cutset* Σ_{st} is the set of all hyperedges $a \in A$ with $T_a \subseteq V_s$ and $h_a \in V_t$. The *size* or *cardinality* of a cut C_{st} is the cardinality of its cutset Σ_{st}, which might be zero.

In standard digraphs, s-t-cuts and s-t-paths are strongly connected to each other. For example, it holds that the minimum cardinality of an s-t-path in a digraph is equal to the maximum number of disjoint s-t-cutsets. In hypergraphs, there are only much weaker relations between the number and cardinality of paths and the number and cardinality of cutsets. We want to state one of the results found in the related literature, which should provide a first insight into the subtle relations between cuts and paths in hypergraphs.

Theorem 1.2.6 (Gallo, Longo, Pallottino, and Nguyen (1993)). *In a directed*

hypergraph $H = (V, A)$ *the following two inequalities hold:*

$$\begin{aligned} & \min\{|\Pi_{st}| \mid \Pi_{st} \textit{ is a hyperpath from } s \textit{ to } t\} \\ \geq\ & \textit{maximum number of disjoint } s\textit{-}t\textit{-cutsets} \\ \geq\ & \min\{|P_{st}| \mid P_{st} \textit{ is an } s\textit{-}t\textit{-path}\}. \end{aligned}$$

$$\begin{aligned} & \textit{maximum number of disjoint } s\textit{-}t\textit{-paths} \\ \geq\ & \min\{|\Sigma_{st}| \mid \Sigma_{st} \textit{ is an } s\textit{-}t\textit{-cutset}\} \\ \geq\ & \textit{maximum number of disjoint hyperpaths from } s \textit{ to } t. \end{aligned}$$

Of course, strict inequality may hold in all cases. For an example and the proofs, we refer to Gallo et al. (1993). In Chapter 2, we will consider similar structures on AND/OR-networks.

We close this chapter with a short overview over some of the work on hypergraphs found in the literature. An early work on general directed hypergraphs noted above is Gallo et al. (1993). The authors investigate paths, hyperpaths, and connectivity matters on directed hypergraphs, as well as cuts and cutsets. Several hypergraph traversals are introduced. In addition, weighted hypergraphs are considered and various applications of directed hyperpaths are discussed. Noteworthy later papers by members of the same research group are Gallo, Gentile, Pretolani, and Rago (1998), Cambini, Gallo, and Scutellá (1997), and Gallo and Scutellá (1999). The latter paper deals with minimum cost hyperpaths and hyperflows and their computation. The second part is devoted to an overview over several applications of directed hypergraphs, such as formal languages, relational databases, production and manufacturing systems, and public transportation models. In Ausiello, Giaccio, Italiano, and Nanni (1992) and in Ausiello, Italiano, and Nanni (1998) the focus is laid on complexity and computation of minimal directed hyperpaths according to different cost measures. For some of the cost measures, the problem of finding a minimal hyperpath is proved to be NP-hard, whereas for other measures, polynomial time algorithms are presented in Ausiello et al. (1992). In addition, dynamic maintenance of optimal hyperpaths is considered. In Ausiello et al. (1998), an analysis of the different cost functions is emphasised to provide some insight into the intrinsic complexity of finding optimal hyperpaths. We will be confronted with similar problems later on.

CHAPTER 2

CRITICALITY IN AND/OR-NETWORKS

After the introduction into the subject in Chapter 1, we consider the problem of scheduling a set of AND/OR precedence constrained jobs with strictly positive processing times such that the makespan is minimized, short $\circ|ao\text{-}prec|C_{\max}$. In Chapter 1 we saw that this problem is solved by the earliest start schedule which can be computed efficiently with a Dijkstra-like algorithm. Nevertheless, in project management not only an optimal solution is of practical interest but also an analysis of the behaviour of this solution when certain input parameters change. In a real project, various parameters of the instance can change. A machine can break down or a worker get ill. A job can take longer (or shorter) due to a number of external influences like the weather or material matters. Some of the jobs in a project are not critical in the sense that, if they take longer, they do not increase the makespan. Whereas other jobs may directly influence the project completion time. Most of the time any delays in the project completion involve high costs so that it might be better to invest some money into the prolonged critical job, to keep its processing time fixed. In any case, a thorough analysis of the problem structure can provide the project planner with a useful tool to handle any external influences. In this chapter we want to give such an analysis, a qualitative and a quantitative one, for AND/OR-networks. We will define critical jobs and sets in AND/OR-networks in Section 2.2. Thereafter, we will analyse their structural properties and treat complexity matters. It will turn out that there is a close relation between critical sets and fulfilling assignments for monotone Boolean functions, thus providing a complexity result in the theory of monotone Boolean functions. Afterwards we will turn to the quantitative analysis. We examine the change of the makespan when the processing times of critical sets change and state a previously known complexity result for the time-cost trade-off problem in AND/OR-networks.

2.1 Critical Jobs in Project Networks

To introduce the reader into the concept of criticality in a project network, we will present the different notions of criticality in standard precedence digraphs in

this section.

Loosely speaking, a job is called critical if its processing time has a direct impact on the objective function, in our case the makespan. In standard project networks one can distinguish three conceptually different but equivalent notions of criticality. Given a precedence digraph $G = (V, E)$ with strictly positive processing times and an earliest start schedule ES respecting the precedence constraints represented by E, we consider the following notions of criticality.

- Job $j \in V$ is critical if any increase in its processing time results in an increase of the makespan. Thus, a critical job is able to delay the completion of the whole project.

- Job j is critical if it belongs to a longest path, that is an inclusion-minimal set $P \subseteq V$ with the property that an earliest start schedule for the subproject induced by P has the same makespan as the whole project. In other words, the jobs on a longest path determine or preserve the makespan when the processing times of other jobs are decreased.

- Job j is critical, if it is contained in an inclusion-minimal set $C \subseteq V$ with the property that any decrease in the processing times of all jobs in C results in a decrease of the makespan. Such a set C induces a cut-set, that is a set having a non-empty intersection with every longest path in the network.

These concepts of criticality are well known to be equivalent for standard precedence digraphs. They are common and useful tools in project management. See for example Möhring and Radermacher (1989) for an overview of techniques in and applications of project analysis.

If we assume the digraph G to have a common source s and a common sink t like we have introduced them for the AND/OR-network N in Section 1.1.1, the longest paths in a conventional network G are precisely the tight s-t-paths, that is s-t-paths only consisting of tight edges. Determining a critical job, a longest path, or the network of all jobs on longest paths can easily be done by standard algorithms in polynomial time.

2.2 Critical Jobs in AND/OR-Networks

Inspired by the ideas of criticality and motivated by their wide usefulness in theory and applications, we will discuss how these notions of criticality can be carried over to AND/OR-networks. We state four definitions for a job in an AND/OR-network to be critical. Three of them correspond to the three different concepts of criticality mentioned above and the last one corresponds to the set

or the "bulk" of all critical jobs. It will turn out that, in contrast to the standard project network case, not all four of them are equivalent in general.

Let a problem instance (N, p) be given, where $N = (V \cup W, E)$ is a feasible AND/OR-network and $p = (p_1, \ldots, p_n)$ is a vector of strictly positive processing times on the jobs in V. Consider the unique earliest start schedule $S = (S_1, \ldots, S_n)$ and the corresponding vector of completion times $C = (C_1, \ldots, C_n)$. The makespan of schedule S is denoted by $C_{\max}(S)$. We will consider modified processing time vectors. Let $U \subseteq V$ be some set of jobs and $0 < \varepsilon < p_{\min}$ be a small constant. We define $p^{[U+\varepsilon]}$ to be the processing time vector where the processing times of the jobs in U are increased by ε, that is

$$p_j^{[U+\varepsilon]} := \begin{cases} p_j + \varepsilon & \text{if } j \in U, \\ p_j & \text{otherwise.} \end{cases}$$

Similarly, $p^{[U-\varepsilon]}$ denotes the processing time vector where we decrease the processing times of the jobs in U by ε. The earliest start time vector of $(N, p^{[U+\varepsilon]})$ shall be denoted by $S^{[U+\varepsilon]}$ and its makespan by $C_{\max}(S^{[U+\varepsilon]})$.

Now consider the impact of a decrease in the processing times of a certain set of jobs.

Definition 2.2.1. *A set $U^C \subseteq V$ of jobs is called* cut-critical *in (N, p) if it is inclusion-minimal with respect to property*

(C) $\quad \forall\, 0 < \varepsilon < p_{\min} : \quad C_{\max}(S^{[U^C-\varepsilon]}) < C_{\max}(S).$

Job $j \in V$ is cut-critical *if there exists a cut-critical set U^C with $j \in U^C$.*

Definition 2.2.2. *A set $U^P \subseteq V$ of jobs is called* path-critical *in (N, p) if it is inclusion-minimal with respect to property*

(P) $\quad \exists\, 0 < \varepsilon < p_{\min} : \quad C_{\max}(S^{[V\setminus U^P-\varepsilon]}) = C_{\max}(S).$

Job $j \in V$ is path-critical *if there exists a path-critical set U^P with $j \in U^P$.*

We can define another two versions of criticality in the same way by looking at the impact of an increase in the processing times of a set of jobs.

Definition 2.2.3. *A set $U^D \subseteq V$ of jobs is called* delay-critical *in (N, p) if it is inclusion-minimal with respect to property*

(D) $\quad \forall\, \varepsilon > 0 : \quad C_{\max}(S^{[U^D+\varepsilon]}) > C_{\max}(S)$

Job $j \in V$ is delay-critical *if there exists a delay-critical set U^D with $j \in U^D$.*

Definition 2.2.4. *A set $U^B \subseteq V$ of jobs is called* bulk-critical *in (N, p) if it is inclusion-minimal with respect to property*

(B) $\exists\, \varepsilon > 0 : \quad C_{\max}(S^{[V \setminus U^B + \varepsilon]}) = C_{\max}(S)$

Job $j \in V$ is bulk-critical *if there exists a bulk-critical set U^B with $j \in U^B$.*

Before we investigate the meaning of the definitions, we want to make two remarks. In Section 1.2.2 we have defined an edge (u, v) to be tight in schedule S for (N, p) if $S_v = S_u + p_u$.

Remark 2.2.5. *For the path-critical and bulk-critical sets, the existence of an $\varepsilon > 0$ fulfilling property (P) and (B), respectively, implies that property (P) and (B) hold for all $0 < \delta \leq \varepsilon$ due to the monotonicity of the makespan.*

Remark 2.2.6. *For cut-critical and delay-critical sets, the term $\forall\, \varepsilon > 0$ in property (C) and (D), respectively, ensures that only tight edges are taken into account.*

To illustrate the significance of Remark 2.2.6 we give a little example for the delay-critical case in Figure 2.1, where $0 < \alpha < 1$ is some small constant. By definition, job 1 is delay-critical, as every little increase in its processing time implies a delay of the start time of job 3 and thus an increase in the makespan. Job 2, in contrast, is not critical. Of course there exists an $\varepsilon > 0$, namely any $\varepsilon > \alpha$, such that an increase of p_2 by ε implies an increase of the makespan, but this does not hold for all $\varepsilon > 0$. Obviously, this is what we want, as job 2 is not critical in the standard network case, and in fact, Figure 2.1 presents just a standard precedence digraph.

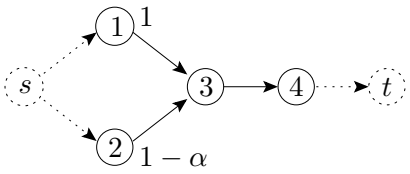


Figure 2.1: Examples for critical and non-critical jobs.

Remark 2.2.6 already indicates that the tight subnetwork N^{tight} of N plays a special role. For several of the following observations it will be crucial to prevent non-tight edges from becoming tight by changing processing times of jobs. This can be done by the right choice of the amount of change. If we denote the set of tight edges in a schedule S by $E^{tight}(S)$ we can relate the sets of tight edges with respect to different processing time schedules as follows.

Lemma 2.2.7. *Let $(N = (V \cup W, E), p)$ be a scheduling instance and $U \subseteq V$ a set of jobs. Then*

$$E^{tight}(S^{[U-\varepsilon]}) \subseteq E^{tight}(S) \;\; \textit{and} \;\; E^{tight}(S^{[U+\varepsilon]}) \subseteq E^{tight}(S)$$

for all ε with

$$0 < \varepsilon < \frac{slack(N,p)}{|U|}.$$

In words this means that if ε is small enough, where small enough depends on the slack of (N,p), we can either increase or decrease the processing times of an arbitrary set of jobs by ε and the set of edges that are tight in the new schedule will be a subset of the tight edges in the original schedule. The proof is quite straightforward.

Proof. Let $U \subseteq V$ be a set of jobs and suppose we increase or decrease the processing time of every job in U by some ε with $0 < \varepsilon < slack(N,p)/|U|$. The increase or decrease of the processing time of one job $j \in U$ by any positive ε can increase, respectively decrease, the start time of any of its successors by at most ε. For any edge with a strictly positive slack $slack(N,p,e)$ it follows that

$$\begin{aligned} slack(N, p^{[U-\varepsilon]}, e) &\geq slack(N,p,e) - |U| \cdot \varepsilon \\ &\geq slack(N,p) - |U| \cdot \varepsilon \\ &> slack(N,p) - |U| \cdot (slack(N,p)/|U|) \\ &\geq slack(N,p) - slack(N,p) = 0 \end{aligned}$$

This shows that a non-tight edge in (N,p) has also a strictly positive slack in $(N, p^{[U-\varepsilon]})$ and therefore it is still non-tight. The same computation can be performed for an increase in the processing time, thus proving the lemma. □

Corollary 2.2.8. *Let $(N = (V \cup W, E), p)$ be a scheduling instance and $U \subseteq V$ a set of jobs. Then*

$$E^{tight}(S^{[U-\varepsilon]}) \subseteq E^{tight}(S^{[U-\varepsilon']}) \text{ and } E^{tight}(S^{[U+\varepsilon]}) \subseteq E^{tight}(S^{[U+\varepsilon']})$$

for all ε and ε' with

$$0 < \varepsilon' < \varepsilon < \frac{slack(N,p)}{|U|}.$$

Proof. Let $U \subseteq V$ be a set of jobs and suppose we increase or decrease the processing time of every job in U by some ε and by some ε' with $0 < \varepsilon' < \varepsilon < slack(N,p)/|U|$. By the same argument as in the proof of Lemma 2.2.7, we get that $slack(N,p) \leq slack(N, p^{[U-\varepsilon']}) + |U| \cdot \varepsilon'$. It follows that

$$\varepsilon < \frac{slack(N,p)}{|U|} \leq \frac{slack(N, p^{[U-\varepsilon']}) + |U| \cdot \varepsilon'}{|U|} \leq \frac{slack(N, p^{[U-\varepsilon']})}{|U|} + \varepsilon'.$$

By the choice of $\varepsilon' < \varepsilon$, we can apply Lemma 2.2.7 with $(N, p^{[U-\varepsilon']})$ instead of (N,p) and set U for $0 < \varepsilon - \varepsilon' < slack(N, p^{[U-\varepsilon']})/|U|$ and get the desired result. The case of increased processing times can be shown accordingly. □

Let us come back to the critical jobs now. We want to have a closer look on the four definitions and relate them to the three notions of criticality in standard networks given in Section 2.1. To do so, we will apply the definition to a pure AND-network G. First consider the definition of a cut-critical job. If job j is in some inclusion-minimal set U^C fulfilling property (C), it is easy to see that U^C has to contain one job of each longest path in G, thus inducing a cut-set through all longest paths of G. So, Definition 2.2.1 corresponds to the third notion of criticality mentioned in Section 2.1. Now consider Definition 2.2.2 of a path-critical job. This definition directly implies that U^P is a longest path in an AND-network G. If we apply Definition 2.2.3 to digraph G, it is easy to see that the set U^D just contains j itself, in direct correspondence of the first notion of criticality in Section 2.1. It is left to consider Definition 2.2.4. Here we are looking for an inclusion-minimal set U^B such that we can increase the processing time of every job not in U^B without increasing the makespan. In a standard network every single critical job has the property that an increase in its processing time implies an increase of the makespan. Thus, U^B has to contain every critical job in G and no other job. We see that, in an AND-network G, there is exactly one bulk-critical set, namely the set of all critical jobs.

Applied to an AND-network G, the four given definitions are equivalent to the standard concept of criticality. As the names should indicate, a cut-critical set induces a cut-set through all longest paths, a path-critical set corresponds to a longest path, a delay-critical set corresponds to one critical job, and a bulk-critical set corresponds to the set of all critical jobs. It is helpful to have those intuitions in mind, when dealing with the four definitions.

2.3 Structural Properties of Critical Jobs and Sets

We will now investigate the structural properties of the different definitions of a critical job and set in an AND/OR-network more formally. As we have already mentioned, it will turn out that they are not equivalent in a general AND/OR-network. Nevertheless, they form two pairs of equivalent notions, as we will see in this section.

2.3.1 Clutters and Blocking Clutters

To be able to classify the critical sets and point out their structural properties, we have to introduce clutters and blocking clutters first. We will restrict our presentation to the concepts needed in the thesis. In Fulkerson (1971) an easy introduction to the theory of clutters is given, including everything that will be presented here. A survey on clutters and blocking clutters in the context of duality

of independence systems can be found in Korte and Vygen (2000) for example. We have modified the notation used in the textbooks slightly to fit our purpose.

Given a finite set V and some $\mathcal{X} \subseteq 2^V$ the system $(V, \mathcal{X})$ is called a *clutter*, if $X \not\subseteq Y$ for all $X, Y \in \mathcal{X}$ with $X \neq Y$. The *blocking clutter* of a clutter $(V, \mathcal{X})$ is defined as

$$BL(V, \mathcal{X}) := (V, \{Y \subseteq V \mid Y \cap X \neq \emptyset \ \forall X \in \mathcal{X}, Y \text{ minimal with this property}\}) .$$

Observation 2.3.1. *For a pair of blocking clutters $(V, \mathcal{X})$ and $(V, \mathcal{Y}) = BL(V, \mathcal{X})$ the following holds.*

1. $BL(V, \mathcal{Y}) = (V, \mathcal{X})$
2. $\bigcup_{X \in \mathcal{X}} X = \bigcup_{Y \in \mathcal{Y}} Y$

Blockers appear in a number of natural settings, we give a well known graph theoretic example here. Consider an undirected graph $G = (V, E)$ and the set of spanning trees $\mathcal{X}$ of G. Then $(E, \mathcal{X})$ is a clutter and $(E, \mathcal{Y})$, with $\mathcal{Y}$ the set of minimal cut sets, is the blocking clutter of $(E, \mathcal{X})$. This and some other easy graph theoretic examples are stated in Korte and Vygen (2000).

A pair of blocking clutters $(V, \mathcal{Y})$ and $(V, \mathcal{X})$ has the *Max-Flow-Min-Cut property* if, for all cost functions $c : V \to \mathbb{R}_+$,

$$\min\{c(X) \mid X \in \mathcal{X}\} = \max\left\{ \sum_{Y \in \mathcal{Y}} f_Y \;\middle|\; f \in \mathbb{R}_+^{\mathcal{Y}}, \sum_{Y \in \mathcal{Y}: v \in Y} f_Y \leq c(v) \ \ \forall v \in V \right\},$$

where $c(X) = \sum_{v \in X} c(v)$. In the literature, this property is defined in different ways. The definition presented here can be found in Fulkerson (1971) and Lehman (1979), for example. The Max-Flow-Min-Cut property provides a strong duality result for a pair of blocking clutters and thereby can reduce the complexity of a number of problems on the clutters.

2.3.2 Systems of Critical Sets

Now, let us come back to the critical jobs and sets in AND/OR-networks. We need some more notation. Given an AND/OR-network $N = (V \cup W, E)$ with processing time vector p, let

$$\begin{aligned}
\mathcal{C} &:= \{U^C \mid U^C \text{ cut-critical set for } (N, p)\}, \\
\mathcal{P} &:= \{U^P \mid U^P \text{ path-critical set for } (N, p)\}, \\
\mathcal{D} &:= \{U^D \mid U^D \text{ delay-critical set for } (N, p)\}, \text{ and} \\
\mathcal{B} &:= \{U^B \mid U^B \text{ bulk-critical set for } (N, p)\}
\end{aligned}$$

be the sets of cut-critical, path-critical, delay-critical, and bulk-critical sets of (N, p), respectively. We will consider the different systems $(V, \mathcal{C})$, $(V, \mathcal{P})$, $(V, \mathcal{D})$, and $(V, \mathcal{B})$ of critical sets. By definition, a set $U^C \in \mathcal{C}$ is inclusion-minimal with respect to property (C) and therefore $U_1^C \not\subseteq U_2^C$, for any two sets $U_1^C, U_2^C \in \mathcal{C}$ with $U_1^C \neq U_2^C$. The same argument holds for the other three set systems, thus proving the next lemma.

Lemma 2.3.2. *The systems $(V, \mathcal{C})$, $(V, \mathcal{P})$, $(V, \mathcal{D})$, and $(V, \mathcal{B})$ are clutters.*

This was an easy observation. The next theorem tells us something more about the relations between pairs of set systems.

Theorem 2.3.3. *For $(N = (V \cup W, E), p)$, the system $(V, \mathcal{C})$ is the blocking clutter of $(V, \mathcal{P})$.*

Proof. We have to prove that each set $U^C \in \mathcal{C}$ has a nonempty intersection with every set $U^P \in \mathcal{P}$ and that every element in the blocking clutter of $\mathcal{P}$ is a cut-critical set.

To achieve a contradiction, suppose that there exists a $U^C \in \mathcal{C}$ such that for some $U^P \in \mathcal{P}$, $U^C \cap U^P = \emptyset$. Then, U^C is contained completely in the complement of U^P. Since U^P is path-critical, it has property (P), that is the makespan remains unchanged, even if the processing times of all jobs in its complement are decreased by some small amount. This contradicts property (C) of the cut-critical set U^C, which is part of the complement of U^P. Thus, every set $U^C \in \mathcal{C}$ has to have at least one element in common with every set $U^P \in \mathcal{P}$. It remains to show that the cut-critical sets are all elements in the blocking clutter of $(V, \mathcal{P})$.

Again for contradiction, suppose that that there exists a set $Y \in BL(V, \mathcal{P})$ in the blocking clutter of $(V, \mathcal{P})$ that is not cut-critical. By definition of the blocking clutter, $Y \cap U^P \neq \emptyset$ for all path-critical sets $U^P \in \mathcal{P}$. Thus $V \setminus Y$ does not contain a complete path-critical set which implies that the makespan decreases if the processing times of all jobs in Y are decreased. It follows that Y has property (C) and therefore contains a cut-critical set. By assumption, Y itself is not cut-critical, thus there exists a cut-critical set $U^C \subsetneq Y$. As every cut-critical set is a blocking set itself, this contradicts the minimality of Y and concludes the proof. □

A direct consequence of Theorem 2.3.3 and Point 2 of Observation 2.3.1 is the equivalence of the properties of being path-critical and being cut-critical.

Corollary 2.3.4. *For $(N = (V \cup W, E), p)$ a job $j \in V$ is path-critical if and only if j is cut-critical.*

As we will see in the following, the same argumentation holds for the systems of delay-critical and bulk-critical sets of an AND/OR-network N.

Theorem 2.3.5. *For $(N = (V \cup W, E), p)$, the system $(V, \mathcal{D})$ is the blocking clutter of $(V, \mathcal{B})$.*

Proof. We have to show that each set $U^D \in \mathcal{D}$ has a nonempty intersection with every set $U^B \in \mathcal{B}$ and that every set in $BL(V, \mathcal{B})$ is delay-critical.

To achieve a contradiction, suppose that there exist a $U^D \in \mathcal{D}$ such that for some $U^B \in \mathcal{B}$, $U^D \cap U^B = \emptyset$. Then, U^D is contained completely in the complement of U^B. Since U^B is bulk-critical, it has property (B): the makespan remains unchanged, even if the processing times of all jobs in its complement are increased by some small amount. This contradicts property (D) of the delay-critical set U^D, which is part of the complement of U^B. Thus, every set $U^D \in \mathcal{D}$ has to have at least one element in common with every set $U^B \in \mathcal{B}$. It remains to show that every set in the blocking clutter of $(V, \mathcal{B})$ is delay-critical.

Consider a set $Y \in BL(V, \mathcal{B})$, then $Y \cap U^B \neq \emptyset$ for all $U^B \in \mathcal{B}$. It follows that there does not exist a bulk-critical set in the complement of Y and therefore the makespan increases if the processing times of all jobs in Y are increased. So Y has property (D). This means that either Y is delay-critical or contains a proper subset that is delay-critical. As we have proved that every delay-critical set is in $BL(V, \mathcal{B})$, the existence of a proper subset of Y which is delay-critical yields a contradiction to the minimality of the clutter element Y. □

A direct consequence of Theorem 2.3.5 and Point 2 of Observation 2.3.1 is the following corollary.

Corollary 2.3.6. *For $(N = (V \cup W, E), p)$ a job $j \in V$ is bulk-critical if and only if j is delay-critical.*

We have proved that the four definitions of criticality form two pairs related through the blocking property of clutters. Now, we want to show that not all four definitions are equivalent. Therefore consider the two networks in Figure 2.2 each with a vector of unit processing times.

Consider the three sets $U_1 = \{1, 3, 5\}$, $U_2 = \{1, 2, 4, 5\}$, and $U_3 = \{1, 3, 4, 5\}$. In the left network N^a, U_1 and U_2 are the path-critical sets, thus every vertex in $V = \{1, \ldots, 5\}$ is path-critical. Set U_3 is the only bulk-critical set of network N^a, which can be seen as follows. To preserve the makespan against an increase in processing times, we may not change the completion time of job 5, which implies that we may neither increase its processing time and therefore $5 \in U_3$, nor its start time. To preserve the start time of job 5, the completion times of job 3 and 4 may not change, implying that $3, 4 \in U_3$. To preserve the start time of job 3, the completion time of job 1 may not change, thus $1 \in U_3$. To preserve the start time of job 4, the completion time of the OR-node w may not change. This is achieved by either fixing the completion time of job 1 or job 2. Job 1 is already

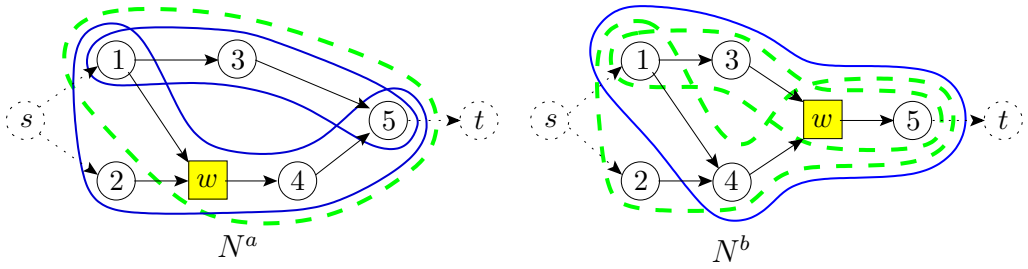


Figure 2.2: Two AND/OR-networks N^a and N^b with processing time vectors $p^a = p^b = (1, \ldots, 1)$. The closed thin curves mark the path-critical sets, the dashed curves mark the bulk-critical sets. In N^a job 2 is path-critical but not bulk-critical, whereas in N^b job 2 is bulk-critical but not path-critical

in U_3 because of job 3. By the inclusion-minimality of bulk-critical set U_3, job 2 cannot be in U_3. There is no other possibility to get a bulk-critical set, thus 2 is not bulk-critical. In the right network N^b, it is the other way around. The sets U_1 and U_2 are bulk-critical and U_3 is the only path-critical set of the network by the same arguments as above, implying that job 2 is not path-critical.

2.3.3 The Max-Flow-Min-Cut Property of Critical Sets

If pairs of blocking clutters have the Max-Flow-Min-Cut property, it is likely that there exists a polynomial time algorithm to compute a minimum element of the blocking clutter. Consider a pair of blocking clutters $(V, \mathcal{Y})$ and $(V, \mathcal{X})$ that have the Max-Flow-Min-Cut property, some cost function and an integer $k > 0$. If there exists a polynomial time algorithm to verify that a set is in the clutter, respectively in the blocking clutter, then the decision problem, "does there exists a set $X \in \mathcal{X}$ with costs at most k" is in NP∩coNP.

We will again first consider the blocking clutter pair of path-critical and cut-critical sets. Unfortunately, in general, $(V, \mathcal{P})$ and $(V, \mathcal{C})$ do not have the Max-Flow-Min-Cut property.

Consider the AND/OR-network in Figure 2.3 for a counter-example. It is easy to check that $U_1^P = \{1, 2, 4\}$, $U_2^P = \{1, 3, 5\}$, and $U_3^P = \{2, 3, 6\}$ are the path-critical sets in $\mathcal{P}$, while $\{1, 2\}$, $\{1, 3\}$, $\{2, 3\}$, $\{1, 6\}$, $\{2, 5\}$, $\{3, 4\}$, and $\{4, 5, 6\}$ are the cut-critical sets in $\mathcal{C}$. Let $c(j) = 1$ for all nodes $j \in V$. With this cost function c, every minimum cut-critical set $U^C \in \mathcal{C}$, for example $\{1, 3\}$, has a value of $c(U^C) = 2$. On the other hand, the value of a maximum flow on the set

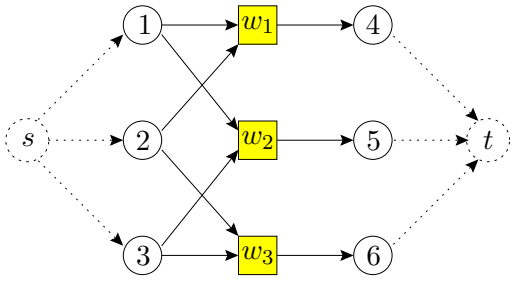


Figure 2.3: $(V, \mathcal{P})$ and $(V, \mathcal{C})$ do not have the Max-Flow-Min-Cut property for the presented network N with $p = (1, \ldots, 1)$.

of paths $\mathcal{P}$ under cost function c can be bounded from above as follows:

$$\begin{aligned}
&\max\{ \sum_{U^P \in \mathcal{P}} f_{U^P} \mid f \in \mathbb{R}_+^{\mathcal{P}}, \sum_{U^P \in \mathcal{P}: v \in U^P} f_{U^P} \leq c(v)\ \forall v \in V\} \\
\leq\ &\max\{ \sum_{U^P \in \mathcal{P}} f_{U^P} \mid f \in \mathbb{R}_+^{\mathcal{P}}, \sum_{U^P \in \mathcal{P}: v \in U^P} f_{U^P} \leq c(v)\ \forall v \in \{1,2,3\}\} \\
\leq\ &\max\{ \sum_{U^P \in \mathcal{P}} f_{U^P} \mid f \in \mathbb{R}_+^{\mathcal{P}}, \sum_{U^P \in \mathcal{P}} 2 f_{U^P} \leq 3\} \leq \frac{3}{2}
\end{aligned}$$

This simple little example shows that there is a gap between a maximum flow on the set of path-critical sets and the minimum costs of a cut-critical set in a general AND/OR-network. A natural question arising is to ask, whether this gap can be bounded. A negative answer will be given by the next theorem. We need to introduce some notation beforehand.

For a given instance $(N = (V \cup W, E), p)$ we consider the ratio $\varrho(N)$ of the minimum value of a cut-critical set over a maximum flow on the system of path-critical sets of network N:

$$\varrho(N) := \frac{\min\{c(U^C) \mid U^C \in \mathcal{C}\}}{\max\{\sum_{U^P \in \mathcal{P}} f_{U^P} \mid f \in \mathbb{R}_+^{\mathcal{P}},\ \sum_{U^P \in \mathcal{P}: v \in U^P} f_{U^P} \leq c(v) \text{ for all } v \in V\}}.$$

We define the *Max-Flow-Min-Cut gap* $\varrho(n)$ to be the maximal ratio $\varrho(N)$ of all AND/OR-networks N on n jobs, that is

$$\varrho(n) = \max\{\, \varrho(N) \mid N = (V \cup W, E) \text{ with } |V| = n \,\}.$$

We are able to prove the following lower bound on the Max-Flow-Min-Cut gap of AND/OR-networks.

Theorem 2.3.7. *Let $\varrho(n)$ be the Max-Flow-Min-Cut gap for AND/OR-networks on n jobs, then*

$$\varrho(n) \in \Omega(\log n).$$

In standard acyclic graphs, the Max-Flow-Min-Cut property holds and it is possible to compute minimum cuts efficiently. This is the main reason for the time-cost trade-off problem with standard precedence constraints to be efficiently solvable. The Max-Flow-Min-Cut gap in AND/OR-networks already indicates that the time-cost trade-off problem for AND/OR precedence constrained jobs might be a hard problem. We will see in Section 2.6.2 that this is indeed the case.

The proof of the Max-Flow-Min-Cut gap will be done by an encoding of a SET COVER instance into an AND/OR-network such that the integrality gap result of the SET COVER problem carries over to our Max-Flow-Min-Cut gap problem. The integrality gap is the ratio between the optimal integral solution over the optimal fractional solution of the linear programming formulation of the problem.

> SET COVER: An instance of (weighted) SET COVER is given by a finite ground set $V = \{1, \ldots, n\}$, a set $\mathcal{S} = \{S_1, \ldots, S_m\}$, $S_i \subseteq V$ for $i = 1, \ldots, m$, of subsets of V with $\bigcup_{S_i \in \mathcal{S}} S_i = V$, and a cost function $c > 0$ for the sets in $\mathcal{S}$. The problem is to find a set $\mathcal{S}' \subseteq \mathcal{S}$ covering all elements in V with minimal costs, that is $\bigcup_{S_i \in \mathcal{S}'} S_i = V$ and $\sum_{S_i \in \mathcal{S}'} c(S_i)$ minimal.

For the description of the SET COVER problem we will follow the notation of Vazirani (2001) and omit any proofs given there.

Proof. For an integer $n > 0$, let $n' = 2^k - 1 \leq n < 2^{k+1} - 1$, where k is a positive integer. The SET COVER instance under consideration has ground set $U = \{x_1, \ldots, x_{n'}\}$. Collect the coefficients of the binary expansions of each i, $1 \leq i \leq n'$, as a k-dimensional vector over the two-element Galois field $GF[2]$. Let $\mathbf{i}$ denote this vector and $\mathbf{i} \cdot \mathbf{j}$ the inner product of the vectors $\mathbf{i}$ and $\mathbf{j}$ in $GF[2]$. Now define $S_i := \{\, x_j \mid \mathbf{i} \cdot \mathbf{j} = 1 \,\}$, for all $i = 1, \ldots, n'$, to be the sets of the SET COVER instance. The cost of each set is fixed to 1.

In this instance, each set S_i has cardinality $2^{k-1} = (n'+1)/2$ and each element x_i is contained in $(n'+1)/2$ sets. Therefore an optimal fractional solution for the SET COVER problem is $f_i = 2/(n'+1)$ with costs $2n'/(n'+1)$. On the other hand, each integral SET COVER has to pick at least k of the sets, thus yielding costs of at least $k = \log_2(n'+1)$, see Vazirani (2001) for any details. Therefore, a lower bound on the integrality gap, that is the fraction of the costs of a minimum integral SET COVER over the costs of a minimum fractional SET COVER, achieved in this example is

$$\frac{\log_2(n'+1)}{2n'/(n'+1)} > \frac{1}{2}\log_2 n'.$$

Now we describe how to construct an AND/OR-network $N = (V \cup W, E)$ on n jobs in which the fractional solution will correspond to a maximum flow on the

path-critical sets and the integral solution of the SET COVER problem to the costs of a minimum cut-critical set in the network.

Let $\ell = n - n'$, by definition of n' it holds that $n \leq 2n'$ and $n \geq \ell$. The node set $V \cup W$ will be composed of four different sets, namely the *set nodes* $\mathcal{S}$, the *element nodes* $\mathcal{E}$, the *index nodes* $\mathcal{I}$, and a chain C. where $\mathcal{S} \cup \mathcal{I} \cup C = V$ and $\mathcal{E} = W$. For each set S_i, $1 \leq i \leq n'$, of the SET COVER instance we introduce an AND-node $S_i \in \mathcal{S}$, and for each element $x_i \in U$, an OR-node $x_i \in \mathcal{E}$. The index node set $\mathcal{I}$ contains the AND-nodes $1, 2, \ldots, n'$, corresponding to the indices of the elements in U. The chain C consist of ℓ jobs $v_1, \ldots, v_\ell$. The processing time of each AND-node is equal to 1. We introduce the following arcs in N. A set vertex $S_i \in \mathcal{S}$, has a directed edge to every element node $x_j \in \mathcal{E}$ for which $x_j \in S_i$. We include edges (x_i, i), from each element node x_i to its index node i. The index nodes are all direct predecessors of v_1, thus the edges (i, v_1) for all $i = 1, \ldots, n'$ are in E. The cost function c is equal to 1 for every AND-node in $\mathcal{S} \cup \mathcal{I}$ and equal to k for every $v \in C$.

As an example for the construction, we have a look on the SET COVER instance for $n' = 2^3 - 1 = 7$. The ground set is $U = \{x_1, \ldots, x_7\}$ and the sets of the SET COVER instance are:

$$\begin{array}{lll} S_1 = \{x_1, x_3, x_5, x_7\} & S_4 = \{x_4, x_5, x_6, x_7\} & S_6 = \{x_2, x_3, x_4, x_5\} \\ S_2 = \{x_2, x_3, x_6, x_7\} & S_5 = \{x_1, x_3, x_4, x_6\} & S_7 = \{x_1, x_2, x_4, x_7\} \\ S_3 = \{x_1, x_2, x_5, x_6\} & & \end{array}$$

The resulting AND/OR-network $N = (V \cup W, E)$ for any $n' = 7 \leq n < 15$ is presented in Figure 2.4, where we represent the chain C consisting of $n - 7$ nodes by a single node C. In this network, a path-critical set U^P contains the chain C, one index node j, and all set nodes S_i with $x_j \in S_i$. We are searching for a maximum flow on the path-critical sets. Again we can bound a maximum flow from above by relaxing the side constraints to just a subset of vertices. We get that

$$\begin{aligned} &\max\{\sum_{U^P \in \mathcal{P}} f_{U^P} \mid f \in \mathbb{R}_+^{\mathcal{P}}, \sum_{U^P \in \mathcal{P}: v \in U^P} f_{U^P} \leq c(v)\ \forall v \in V\} \\ \leq\ &\max\{\sum_{U^P \in \mathcal{P}} f_{U^P} \mid f \in \mathbb{R}_+^{\mathcal{P}}, \sum_{U^P \in \mathcal{P}: v \in U^P} f_{U^P} \leq c(v)\ \forall v \in \mathcal{S}\} \\ \leq\ &\max\{\sum_{U^P \in \mathcal{P}} f_{U^P} \mid f \in \mathbb{R}_+^{\mathcal{P}}, \sum_{U^P \in \mathcal{P}} \frac{n'+1}{2} f_{U^P} \leq n'\} \leq \frac{2n'}{n'+1}. \end{aligned}$$

There are three possibilities for the cut-critical sets U^C. They either consist of a set $U_{\mathcal{S}} \subseteq \mathcal{S}$ of set nodes, covering all the elements, represented by the element nodes in $\mathcal{E}$. Then $|U_{\mathcal{S}}| \geq k$, by the definition of the SET COVER instance. Or they

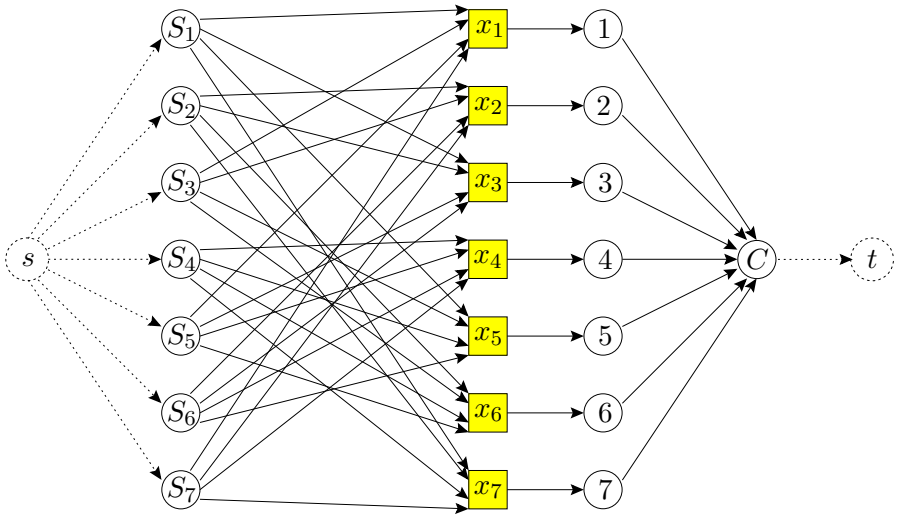


Figure 2.4: AND/OR-network of SET COVER instance for $k = 3$

are composed of a (possibly empty) subset $U_{\mathcal{S}} \subset \mathcal{S}$ of set nodes, $|U_{\mathcal{S}}| < k$, and the set $U_{\mathcal{I}} \subseteq \mathcal{I}$ of the index nodes of the elements not covered by the $S_i \in U_{\mathcal{S}}$. By construction of the network it follows that in these cases, $|U_{\mathcal{I}}| + |U_{\mathcal{S}}| \geq k$ and therefore $|U^C| = |U_{\mathcal{S}} \cup U_{\mathcal{I}}| = |U_{\mathcal{I}}| + |U_{\mathcal{S}}| \geq k = \log_2(n' + 1)$. In addition, every single node in the chain C is a cut-critical set on its own. By definition of the cost function, each job $v \in C$ has costs $c(v) = k = \log_2(n' + 1)$. We now can conclude that for the network N on $n \leq 2n'$ jobs we have

$$\varrho(N) = \frac{\min\{c(U^C)\}}{\max\{\sum_{U^P \in \mathcal{P}} f_{U^P}\}} \geq \frac{\log_2(n'+1)}{2n'/(n'+1)} > \frac{\log_2 n'}{2} \geq \frac{\log_2(n/2)}{2}.$$

For the Max-Flow-Min-Cut property gap then follows

$$\varrho(n) = \max\{\, \varrho(N) \mid N = (V \cup W, E),\ |V| = n \,\} \geq \frac{\log_2(n/2)}{2} = \frac{\log_2 n - 1}{2}$$

which proves the logarithmic lower bound. □

Now consider the systems of bulk-critical and delay-critical sets. Again, in general $(V, \mathcal{B})$ and $(V, \mathcal{D})$ do not have the Max-Flow-Min-Cut property. In Figure 2.5 a network $N = (V \cup W, E)$ with unit processing time vector is given. This network is closely related to the one presented in Figure 2.3 for the pair of path- and cut-critical set systems. We have three bulk-critical sets, $U_1^B = \{1, 2, 4, 7\}$, $U_2^B = \{1, 3, 5, 7\}$, and $U_3^B = \{2, 3, 6, 7\}$, and the delay-critical sets $\{1, 2\}$, $\{1, 3\}$, $\{2, 3\}$, $\{1, 6\}$, $\{2, 5\}$, $\{3, 4\}$, $\{4, 5, 6\}$, and $\{7\}$. Let the cost function c be such

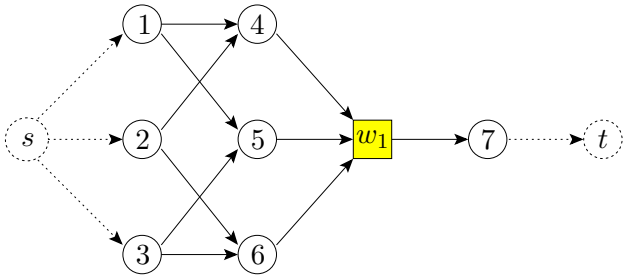


Figure 2.5: $(V, \mathcal{B})$ and $(V, \mathcal{D})$ do not have the Max-Flow-Min-Cut property for network N and $p = (1, \ldots, 1)$.

that $c(j) = 1$ for $j = 1, \ldots, 6$ and $c(7) = 2$. Then the minimal cost of a delay-critical set U^D is $c(U^D) = 2$, while an optimal flow f on the bulk-critical sets is $f_{U_i^B} = 1/2$, for $i = 1, 2, 3$, with a total value of $3/2$.

If we compare the network given in Figure 2.3 with the one presented in Figure 2.5 we see that one emanates from the other by appropriately exchanging AND- and OR-nodes. Let us include the additional AND-node 7 of the network presented in Figure 2.5 into the AND/OR-network given in Figure 2.3 with $p(7) = 1$ and $c(7) = 2$. Then the path-critical sets of the network in Figure 2.3 are exactly the bulk-critical sets of the network in Figure 2.5 and the cut-critical sets of the network in Figure 2.3 correspond to the delay-critical sets of the network in Figure 2.5.

The same construction can be done with the AND/OR-network of Figure 2.4. Replace the element OR-nodes in $\mathcal{E}$ by AND-nodes and remove the set $\mathcal{I}$ of index nodes. Add an additional OR-node w as a direct successor of the element AND-nodes in $\mathcal{E}$ and a direct predecessor of the first node of chain C. Every AND-node is assigned a processing time of 1. The cost function corresponds to the above cost function, that is $c(v) = k$ for all jobs in the chain C and $c(v) = 1$ for all other AND-nodes. With this construction the minimum cost of a delay-critical set U^D is $\min\{c(U^D)\} = k = \log_2(n' + 1)$, where n' is again such that $n' = 2^k - 1 \leq n < 2^{k+1} - 1$. A maximum flow on the bulk-critical sets U^B has a value of $\max\{\sum_{U^B \in \mathcal{B}} f_{U^B}\} \leq 2n'/(n' + 1)$, exactly as in the path-/cut-critical case. This proves that the Max-Flow-Min-Cut gap for the pair of blocking clutters of bulk-critical and delay-critical set systems is also in $\Omega(\log n)$.

Unfortunately, we were not able to prove any upper bound on the Max-Flow-Min-Cut gap. Therefore, among others, the question whether the $\log n$ lower bound is tight, remains open.

We want to close this subsection with a short intuitive explanation of the symmetry property of critical sets used above. Already in the networks of Figure 2.2, a certain symmetric behaviour of path- and bulk-critical sets can be observed. This

behaviour can be explained as follows. Consider a path-critical set U^P and a bulk-critical set U^B, for example. Both of them have to preserve the makespan, when the processing times of the jobs in the complement change. To be able to preserve the makespan against a decrease of processing times, a set U^P has to contain some sort of s-t-path such that for an AND-node $j \in U^P$, U^P also contains (at least) one tight direct predecessor, whereas for an OR-node w occurring on the path, every tight direct predecessor of w has to be in U^P. Now consider a bulk-critical set U^B that has to preserve the makespan against an increase of processing times of jobs. It also has to contain some sort of s-t-path such that for an AND-node $j \in U^B$, every tight direct predecessor of j is in U^B, whereas for an OR-node w occurring on the path, one direct tight predecessor suffices to preserve its start time. In this argumentation, the roles of AND- and OR-nodes are just exchanged, which reflects the symmetry of the two examples. We want to remark that we cannot use this symmetry formally to establish any general results. This is due to our structural assumptions on AND/OR-network stated in Remark 1.1.2 and the different roles of AND- and OR-nodes. Examples for the failure of the symmetry argument can be seen later in the proofs of Theorem 2.4.3 and Theorem 2.5.4.

2.3.4 Invariants of Critical Sets

In this section we have an even closer look onto critical sets and prove some nice properties of them, which will facilitate further investigations. One of those properties is that the start times of the jobs in a critical set do not change with a small change in the processing times of the jobs. We also will present another definition of path- and bulk-critical sets that is based on a completely structural point of view.

In the discussions about the behaviour of path- and bulk-critical sets it was hinted that one might also consider certain OR-nodes as critical. We want to do this formally by defining the closure of path- and bulk-critical sets.

Definition 2.3.8. *Let U^P be a path-critical set in (N, p), then the set*

$$\overline{U^P} := U^P \cup \{w \in W \mid \emptyset \neq ImSucc^{tight}(w) \subseteq U^P \text{ and } ImPred^{tight}(w) \subseteq U^P\}$$

is called the closure *of U^P.*

Similarly but not equivalently we define the closure of a bulk-critical set. It does not make sense to define the closure of cut- or delay-critical sets, as this would somehow include a change of processing times of OR-nodes, which are fixed to zero by definition.

Definition 2.3.9. *Let U^B be a bulk-critical set in (N, p), then the set*

$$\overline{U^B} := U^B \cup \{w \in W \mid \emptyset \neq ImSucc^{tight}(w) \subseteq U^B\}$$

is called the closure *of U^B.*

Note that it depends on both, the immediate predecessors and the immediate successor of an OR-node, whether the OR-node belongs to the closure of a path-critical set. In the bulk-critical case, it only depends on the immediate successor. Obviously the definition of the closure of a bulk-critical set looks very simple. One should expect some additional constraint like $ImPred^{tight}(w) \cap U^B \neq \emptyset$ to be necessary for an OR-node w to belong to the closure. Actually it is possible to prove that this property is automatically given and thus redundant for the definition. We will do this later as it will be much easier to show once we have proved the following two lemmas.

Now that we have extended two of the notions of criticality to OR-nodes even, we want to examine the start times of the jobs in critical sets. The following lemmas will be the base of several other results.

Lemma 2.3.10. *Let U^C [U^D] be a cut-critical [delay-critical] set for network $(N = (V \cup W, E), p)$. Then, there exists a $\delta > 0$ such that $S_j^{[U^C-\varepsilon]} = S_j$ [$S_j^{[U^D+\varepsilon]} = S_j$] for all $j \in U^C$ [$j \in U^D$] and for all $0 < \varepsilon \leq \delta$,*

Proof. First we give the proof for the cut-critical case. Let us recall the definition of a cut-critical set. By Definition 2.2.1, U^C is inclusion-minimal with respect to property

(C) $\forall\, 0 < \varepsilon < p_{\min} :\ C_{\max}(S^{[U^C-\varepsilon]}) < C_{\max}(S)$

From the inclusion minimality of U^C it follows that property (C) does not hold for any proper subset of U^C, that is

$$\exists\, \delta > 0 \text{ such that for all } j \in U^C : C_{\max}(S^{[U^C \setminus \{j\}-\delta]}) = C_{\max}(S).$$

By Lemma 1.2.5 stating the monotonicity of the makespan, it follows that this holds for any $0 < \varepsilon \leq \delta$. We choose

$$0 < \delta^C < \min\{\delta, \frac{slack(N,p)}{|V|}\}$$

and keep this δ^C fixed in the following. By Lemma 2.2.7, this choice of δ^C guarantees that $E^{tight}(S^{[U-\varepsilon]}) \subseteq E^{tight}(S)$ for any $0 < \varepsilon \leq \delta^C$ and for any set of jobs $U \subseteq V$.

Let $0 < \varepsilon \leq \delta^C$ and suppose that $S_j^{[U^C-\varepsilon]} < S_j$ for some $j \in U^C$. Among all jobs in U^C with this property, let k be one with maximal start time S_k. Consider the earliest start schedule $S^{[U^C\setminus\{k\}-\varepsilon]}$. In other words we decrease the processing times of all jobs in U^C except k by ε. By the choice of $\varepsilon \leq \delta^C$ we get that $C_{\max}(S^{[U^C\setminus\{k\}-\varepsilon]}) = C_{\max}(S)$. Now choose

$$\gamma = \min\{\varepsilon, \frac{S_k - S_k^{[U^C-\varepsilon]}}{|U^C|}\}.$$

By assumption, $S_k > S_k^{[U^C-\varepsilon]}$, and also $\varepsilon > 0$, thus $\gamma > 0$. Property (C) of the cut-critical set U^C implies that $C_{\max}(S) > C_{\max}(S^{[U^C-\gamma]})$. To shorten notation let

$$S^\varepsilon = S^{[U^C-\varepsilon]},\ S' = S^{[U^C\setminus\{k\}-\varepsilon]},\ \text{and } S^\gamma = S^{[U^C-\gamma]}$$

with corresponding completion time vectors C^ε, C', and C^γ.

We claim that $C_v^\gamma \geq C'_v$ for all nodes $v \in V \cup W$. By contradiction, we assume that $C_v^\gamma < C'_v$ for some node v. Among all nodes v with $C_v^\gamma < C'_v$, we choose u with minimal start time S_u^γ. For the case that u is not unique, we can choose any u with minimal start time S_u^γ as long as the vertices are incomparable. If vertices with the same minimal start time are comparable, we take the preceding one. We have to distinguish two cases.
If $u \neq k$, then

$$C_u^\gamma \;=\; S_u^\gamma + p_u^\gamma \;\geq\; S_u^\gamma + p'_u \;=\; p'_u + \begin{cases} \max\{C_v^\gamma \mid (v,u) \in E\} & \text{if } u \in V, \\ \min\{C_v^\gamma \mid (v,u) \in E\} & \text{if } u \in W. \end{cases}$$

If $u \in V$, then by the minimal choice of u, we have that for all $(v,u) \in E$ it holds that $C_v^\gamma \geq C'_v$ and thus the right hand side is bounded from below by $S'_u + p'_u = C'_u$, yielding a contradiction. If on the other hand $u \in W$, we also get a contradiction by the following estimate of the completion time of u:

$$\begin{aligned} C_u^\gamma &\geq \min\{C_v^\gamma \mid (v,u) \in E\} \\ &= \min\{C_v^\gamma \mid (v,u) \in E^{\mathit{tight}}(S^\gamma)\} \\ &\geq \min\{C'_v \mid (v,u) \in E^{\mathit{tight}}(S^\gamma)\} \\ &\geq \min\{C'_v \mid (v,u) \in E\} = C'_u \end{aligned}$$

The second inequality holds by the minimal choice of u with respect to S^γ.
It remains to consider the case $u = k$. Since shortening the processing time of one job j by γ cannot decrease the start time of job k by more than γ, we get

$$S_k^\gamma \geq S_k - (|U^C| - 1)\gamma \geq S_k^\varepsilon + \gamma \;=\; S'_k + \gamma\,.$$

The second inequality follows from $\gamma \leq (S_k - S_k^{\varepsilon})/|U^C|$. The equality $S_k^{\varepsilon} = S'_k$ holds as $p_j^{\varepsilon} = p'_j$ for all jobs j in the predecessor set of k excluding k. This yields $C_k^{\gamma} = S_k^{\gamma} + p_k - \gamma \geq S'_k + p_k = C'_k$ which completes the proof of the claim.

Summarising, we get the contradiction

$$C_{\max}(S) \; > \; C_{\max}(S^{[U^C-\gamma]}) \; \geq \; C_{\max}(S^{[U^C\setminus\{k\}-\varepsilon]}) \; = \; C_{\max}(S) \; .$$

This concludes the proof for cut-critical sets. Now let us consider the case of a delay-critical set U^D. We can follow the same construction.

By Definition 2.2.3, U^D is inclusion-minimal with respect to the property

(D) $\forall\, 0 < \varepsilon :\; C_{\max}(S^{[U^D+\varepsilon]}) > C_{\max}(S)$,

From the inclusion minimality of U^D it follows that there exists a $\delta > 0$ with the property that increasing the processing time of all but one job in U^D by δ does not affect the makespan, that is

$$\exists\, \delta > 0 \text{ such that for all } j \in U^D : C_{\max}(S^{[U^D\setminus\{j\}+\delta]}) = C_{\max}(S).$$

We choose

$$0 < \delta^D = \min\{\delta, \frac{slack(N,p)}{|V|}\}$$

and keep this δ^D fixed in the following.

Let $0 < \varepsilon \leq \delta^D$ and suppose that $S_j^{[U^D+\varepsilon]} > S_j$, for some $j \in U^D$. Among all jobs in U^D with this property, let k be one with maximal start time S_k. We increase the processing time of all jobs in U^D except k by ε and consider the resulting start times. By the choice of $\varepsilon \leq \delta^D$, we get that $C_{\max}(S^{[U^D\setminus\{k\}+\varepsilon]}) = C_{\max}(S)$. Let

$$\gamma = \min\{\varepsilon, \frac{S_k^{[U^D+\varepsilon]} - S_k}{|U^D|}\}.$$

By assumption, $S_k < S_k^{[U^D+\varepsilon]}$, and also $\varepsilon > 0$, thus $\gamma > 0$. By property (D) of the delay-critical set U^D we know that $C_{\max}(S) \; < \; C_{\max}(S^{[U^D+\gamma]})$. Again to shorten notation let

$$S^{\varepsilon} = S^{[U^D+\varepsilon]},\; S' = S^{[U^D\setminus\{k\}+\varepsilon]},\; \text{and } S^{\gamma} = S^{[U^D+\gamma]}$$

with corresponding completion time vectors C^{ε}, C', and C^{γ}.

We claim that $C_v^{\gamma} \leq C'_v$ for all nodes $v \in V \cup W$, where $C'_v = S'_v + p'_v$. By contradiction, assume that $C_v^{\gamma} > C'_v$ for some node v. Among all nodes v with $C_v^{\gamma} > C'_v$, we choose u with minimal start time S'_u. If u is not unique and the different vertices are comparable to each other, we always take the preceding

one, otherwise we can choose any u with minimum start time. We again have to distinguish two cases.
If $u \neq k$, then

$$C_u^\gamma \;=\; S_u^\gamma + p_u^\gamma \;\leq\; S_u^\gamma + p'_u \;=\; p'_u + \begin{cases} \max\{C_v^\gamma \mid (v,u) \in E\} & \text{if } u \in V, \\ \min\{C_v^\gamma \mid (v,u) \in E\} & \text{if } u \in W. \end{cases}$$

Again we have to treat the cases u being an AND-node and u being an OR-node separately. If $u \in V$ then for all $(v,u) \in E$ it holds that $C_v^\gamma \leq C'_v$ by the minimal choice of u. Therefore the right hand side is bounded from above by $S'_u + p'_u = C'_u$, yielding a contradiction. If $u \in W$ we get the same contradiction by the following estimate:

$$\begin{aligned} C_u^\gamma &\leq \min\{C_v^\gamma \mid (v,u) \in E\} \\ &\leq \min\{C_v^\gamma \mid (v,u) \in E^{tight}(S')\} \\ &\leq \min\{C'_v \mid (v,u) \in E^{tight}(S')\} = C'_u \end{aligned}$$

The minimal choice of u with respect to schedule S' gives us the third inequality, the other equality and inequalities follow by standard arguments.
It remains to consider the case $u = k$. Since prolonging the processing time of one job j by γ cannot increase the start time of job k by more than γ, we get that

$$S_k^\gamma \leq S_k + (|U^D| - 1)\gamma \leq S_k^\varepsilon - \gamma \;=\; S'_k - \gamma \,.$$

The second inequality follows from $\gamma \leq (S_k^\varepsilon - S_k)/|U^D|$. The equality $S_k^\varepsilon = S'_k$ holds as $p_j^\varepsilon = p'_j$ for all jobs j in the predecessor set of k, except k itself. This yields $C_k^\gamma = S_k^\gamma + p_k + \gamma \leq S'_k + p_k = C'_k$ which completes the proof of the claim.

Recapitulating, we get the contradiction

$$C_{\max}(S) \;<\; C_{\max}(S^{[U^D+\gamma]}) \;\leq\; C_{\max}(S^{[U^D \setminus \{k\}+\varepsilon]}) \;=\; C_{\max}(S)$$

which concludes the proof for delay-critical sets.

Choosing $\delta := \min\{\delta^C, \delta^D\}$ concludes the proof of the lemma. □

The same characterisation can be proven for path- and bulk-critical sets. Actually it can be strengthened to the closures of these sets as we will see later.

Lemma 2.3.11. *Let U^P [U^B] be a path-critical [bulk-critical] set for the network $(N = (V \cup W, E), p)$. Then there exists a $\delta > 0$ such that $S_j^{[V \setminus U^P - \varepsilon]} = S_j$ for all $j \in U^P$ [$S_j^{[V \setminus U^B + \varepsilon]} = S_j$ for all $j \in U^B$] and for all $0 < \varepsilon \leq \delta$.*

Proof. First we consider the path-critical case. To this end let U^P be a path-critical set for (N, p). By Definition 2.2.2, U^P is inclusion-minimal with respect to property

(P) $\exists\, 0 < \varepsilon < p_{\min} :\ C_{\max}(S^{[V\setminus U^P-\varepsilon]}) = C_{\max}(S)$,

As we have already stated in Remark 2.2.5 the existence of an $\varepsilon > 0$ implies that $C_{\max}(S^{[V\setminus U^P-\gamma]}) = C_{\max}(S)$ for all $0 < \gamma \leq \varepsilon$. Let ε fulfill property (P), then we choose

$$0 < \delta^P = \min\{\varepsilon, \frac{slack(N,p)}{|V|}\}$$

and keep this δ^P fixed in the following. Note that this choice of δ^P guarantees that the set of tight edges for any processing time vector changed by less than δ^P is a subset of the tight edges for S.

As U^P is inclusion-minimal with property (P), a decrease of the processing times of all jobs not in U^P and one job in U^P results in a decrease of the makespan. For the sake of contradiction, let $0 < \varepsilon \leq \delta^P$ and suppose that there exists a job $j \in U^P$ with $S_j^{[V\setminus U^P-\varepsilon]} < S_j$. Among all jobs with this property, let $k \in U^P$ be one with maximal start time S_k. If k is not unique we can simply take any such k; due to the strictly positive processing times they cannot be in relation to each other. Now choose

$$\gamma = \min\{\varepsilon, \frac{S_k - S_k^{[V\setminus U^P-\varepsilon]}}{|V \setminus U^P| + 1}\}$$

and let

$$S^\varepsilon = S^{[V\setminus U^P-\varepsilon]} \text{ and } S' = S^{[(V\setminus U^P)\cup\{k\}-\gamma]}$$

with respective completion time vectors C^ε and C' to ease the notation. As $k \in V$, by the inclusion-minimality of a path-critical set, it follows that $C_{\max}(S') < C_{\max}(S)$.

We claim that $C_v^\varepsilon \leq C'_v$ for all nodes $v \in V \cup W$. By contradiction we assume that $C_v^\varepsilon > C'_v$ for some node v. Among all nodes with this property, we choose u with minimal start time S'_u. For the case that there exist several u with minimal start time S'_u and they are in relation to each other, we take the preceding vertex, otherwise we can choose any one of them. We have to consider the two cases that either a) $u \neq k$ or b) $u = k$.

a) If $u \neq k$, then

$$C'_u = S'_u + p'_u \geq S'_u + p_u^\varepsilon = p_u^\varepsilon + \begin{cases} \max\{C'_v \mid (v,u) \in E\} & \text{if } u \in V, \\ \min\{C'_v \mid (v,u) \in E\} & \text{if } u \in W. \end{cases}$$

If $k \neq u \in V$, then for all $(v,u) \in E$ it holds that $S'_v \leq S'_u$. By the minimal choice of u this implies that $C'_v \geq C_v^\varepsilon$ for all $(v,u) \in E$. Note that $S'_v = S'_u$ only in the case that v is an OR-node. Then either $C_v^\varepsilon > C'_v$ in which case we would have preferred v itself, or $C_v^\varepsilon \leq C'_v$. Thus

$$C'_u \geq p_u^\varepsilon + \max\{C'_v \mid (v,u) \in E\} \geq p_u^\varepsilon + S_u^\varepsilon = C_u^\varepsilon,$$

which is a contradiction to our assumption.
Otherwise if $u \in W$ we can make the following estimate.

$$\begin{aligned}\min\{C'_j \mid (j,u) \in E\} &= \min\{C'_j \mid (j,u) \in E^{tight}(S')\} \\ &\geq \min\{C^\varepsilon_j \mid (j,u) \in E^{tight}(S')\} \\ &\geq \min\{C^\varepsilon_j \mid (j,u) \in E\} = S^\varepsilon_u\end{aligned}$$

The first inequality again holds by the minimal choice of u and the strictly positive processing times on jobs: for every tight predecessor j of u in S', it holds that $S'_j < S'_u$ implying that $C^\varepsilon_j \leq C'_j$. Again we get the contradiction to our assumption,

$$C'_u \geq p^\varepsilon_u + \min\{C'_j \mid (j,u) \in E\} \geq p^\varepsilon_u + S^\varepsilon_u = C^\varepsilon_u.$$

b) Now consider the remaining case when $u = k$. Shortening the processing time of an arbitrary job by γ cannot decrease the start time of job k by more than γ. Thus

$$S'_k \geq S_k - |V \backslash U^P|\gamma \geq S^\varepsilon_k + \gamma,$$

where the second inequality follows from $\gamma \leq (S_k - S^\varepsilon_k)/(|V \backslash U^P| + 1)$. This yields $C'_k = S'_k + p_k - \gamma \geq S^\varepsilon_k + p_k = C^\varepsilon_k$ which again gives a contradiction and completes the proof of the claim.

Summarising what we have shown so far, we get the contradiction

$$C_{\max}(S) > C_{\max}(S') \geq C_{\max}(S^\varepsilon) = C_{\max}(S).$$

This concludes the proof for path-critical sets U^P.

Now consider the bulk-critical case and let U^B be a bulk-critical set of (N,p). By Definition 2.2.4, U^B is inclusion-minimal with respect to property

(B) $\exists\, 0 < \varepsilon :\ C_{\max}(S^{[V \backslash U^B + \varepsilon]}) = C_{\max}(S)$,

This time let ε fulfill property (B) and choose

$$0 < \delta^B = \min\{\varepsilon, \frac{slack(N,p)}{|V|}\}$$

and keep this δ^B fixed in the following. By Remark 2.2.5, $C_{\max}(S^{[V \backslash U^B + \varepsilon]}) = C_{\max}(S)$ for all $0 < \varepsilon \leq \delta^B$.

As U^B is inclusion-minimal with property (B) an increase of the processing times of all jobs not in U^B and one additional job in U^B results in an increase of the makespan. Let $0 < \varepsilon \leq \delta^B$ and suppose that there exists a job $j \in U^B$ with $S_j^{[V \backslash U^B + \varepsilon]} > S_j$. Among all jobs with this property, let k be the one with maximal

start time S_k. If there are several k to choose from, we can take any of them as jobs with equal start time cannot be comparable. Now choose

$$\gamma := \min\{\varepsilon, \frac{S_k^{[V\setminus U^B+\varepsilon]} - S_k}{|V\setminus U^B| + 1}\}$$

and let

$$S^\varepsilon = S^{[V\setminus U^B+\varepsilon]} \text{ and } S' = S^{[(V\setminus U^B)\cup\{k\}+\gamma]}$$

with respective completion time vectors C^ε and C' to ease the notation. As $k \in V$, it follows from the inclusion-minimality of a bulk-critical set that $C_{\max}(S') > C_{\max}(S)$.

We claim that $C_v^\varepsilon \geq C'_v$ for all nodes $v \in V \cup W$. By contradiction assume that $C_v^\varepsilon < C'_v$ for some v. Among all nodes with this property, we choose u with minimal start time S_u^ε. If there are several u with minimal S_u^ε, we can take any of them if they are not related to each other. If they are comparable, we take the predecessor. Again we have to consider the two cases that either a) $u \neq k$ or b) $u = k$.
a) If $u \neq k$, then

$$C'_u = S'_u + p'_u \leq S'_u + p_u^\varepsilon = p_u^\varepsilon + \begin{cases} \max\{C'_v \mid (v,u) \in E\} & \text{if } u \in V, \\ \min\{C'_v \mid (v,u) \in E\} & \text{if } u \in W. \end{cases}$$

If $k \neq u \in V$, then for all $(v,u) \in E$ it holds that $S_v^\varepsilon \leq S_u^\varepsilon$. By the minimal choice of u this implies that $C'_v \leq C_v^\varepsilon$ for all $(v,u) \in E$. Note that $S_v^\varepsilon = S_u^\varepsilon$ only in the case that v is an OR-node. Then either $C_v^\varepsilon < C'_v$ in which case we would have preferred v itself, or $C_v^\varepsilon \geq C'_v$. Thus

$$C'_u \leq p_u^\varepsilon + \max\{C'_v \mid (v,u) \in E\} \leq p_u^\varepsilon + S_u^\varepsilon = C_u^\varepsilon,$$

which is a contradiction to our assumption.
Otherwise if $u \in W$ we can make the following estimate.

$$\begin{aligned} \min\{C'_j \mid (j,u) \in E\} &\leq \min\{C'_j \mid (j,u) \in E^{tight}(S^\varepsilon)\} \\ &\leq \min\{C_j^\varepsilon \mid (j,u) \in E^{tight}(S^\varepsilon)\} = S_u^\varepsilon \end{aligned}$$

The second inequality holds by the minimal choice of u and the strictly positive processing times on jobs: for every tight predecessor j of u in S^ε, it holds that $S_j^\varepsilon < S_u^\varepsilon$ implying that $C_j^\varepsilon \geq C'_j$. Again we get the contradiction to our assumption,

$$C'_u \leq p_u^\varepsilon + \min\{C'_j \mid (j,u) \in E\} \leq p_u^\varepsilon + S_u^\varepsilon = C_u^\varepsilon.$$

b) Now consider the remaining case $u = k$. Since prolonging the processing time of a job by γ cannot increase the start time of job k by more than γ, we get that

$$S'_k \leq S_k + |V \backslash U^B|\gamma \leq S^\varepsilon_k - \gamma.$$

The second inequality follows from $\gamma \leq (S^\varepsilon_k - S_k)/(|V \backslash U^B| + 1)$. This yields $C'_k = S'_k + p_k + \gamma \leq S^\varepsilon_k + p_k = C^\varepsilon_k$ which gives a contradiction and completes the proof of the claim.

In summary we get the contradiction

$$C_{\max}(S) < C_{\max}(S') \leq C_{\max}(S^\varepsilon) = C_{\max}(S)$$

which completes the proof for the second case of bulk-critical sets.

Choosing $\delta := \min\{\delta^P, \delta^B\}$ concludes the proof of the lemma. □

Now we consider the closure of path- and bulk-critical sets. It is much easier to prove the invariance of the OR-nodes than for the jobs in the set.

Lemma 2.3.12. *Let U^P [U^B] be a path-critical [bulk-critical] set for the network $(N = (V \cup W, E), p)$ and $\overline{U^P}$ [$\overline{U^B}$] the respective closure. Then, there exists a $\delta > 0$ such that $S_v^{[V\backslash U^P - \varepsilon]} = S_v$ for all $v \in \overline{U^P}$ [$S_v^{[V\backslash U^B + \varepsilon]} = S_v$ for all $v \in \overline{U^B}$] and for all $0 < \varepsilon \leq \delta$.*

Proof. For the proof let $\delta > 0$ fulfill the requirement of Lemma 2.3.11 and let $\overline{U^P}$ be the closure of a path-critical set U^P for (N, p). For the sake of contradiction let $0 < \varepsilon \leq \delta$ and suppose that some OR-node $w \in W \cap \overline{U^P}$ of the closed path-critical set $\overline{U^P}$ has the property that $S_w^{[V\backslash U^P - \varepsilon]} < S_w$. According to the construction in the proof of Lemma 2.3.11, ε is such that $E^{tight}(S^{[V\backslash U^P - \varepsilon]}) \subseteq E^{tight}(S)$. Therefore we can make the following estimate about the start times of w:

$$\begin{aligned} S_w^{[V\backslash U^P - \varepsilon]} &= \min\{S_j^{[V\backslash U^P - \varepsilon]} + p_j^{[V\backslash U^P - \varepsilon]} \mid (j,w) \in E^{tight}(S^{[V\backslash U^P - \varepsilon]})\} \\ &\geq \min\{S_j^{[V\backslash U^P - \varepsilon]} + p_j^{[V\backslash U^P - \varepsilon]} \mid (j,w) \in E^{tight}(S)\} \\ &= \min\{S_j + p_j \mid (j,w) \in E^{tight}(S)\} = S_w, \end{aligned}$$

where the second equality follows by the definition of the closure of path-critical sets together with Lemma 2.3.11. This gives a contradiction to our assumption and proves the path-critical case.

Now let $\overline{U^B}$ be the closure of a bulk-critical set U^B for (N, p). Again assume that there exists an OR-node $w \in W \cap \overline{U^B}$ with the property that $S_w^{[V\backslash U^B + \varepsilon]} > S_w$ for some $0 < \varepsilon \leq \delta$. Consider the start time of its tight direct successor

$j = ImSucc^{tight}(w)$ with respect to S. By the definition of the closure of a bulk-critical set, we know that j exists and that it is in U^B as otherwise w would not be in $\overline{U^B}$.

$$\begin{aligned} S_j^{[V\setminus U^B+\varepsilon]} &= \max\{S_v^{[V\setminus U^B+\varepsilon]} + p_v^{[V\setminus U^B+\varepsilon]} \mid (v,j) \in E\} \\ &\geq \max\{S_v^{[V\setminus U^B+\varepsilon]} + p_v^{[V\setminus U^B+\varepsilon]} \mid (v,j) \in E^{tight}(S)\} \\ &\geq S_w^{[V\setminus U^B+\varepsilon]} + p_w^{[V\setminus U^B+\varepsilon]} = S_w^{[V\setminus U^B+\varepsilon]} > S_w = S_j, \end{aligned}$$

where the last equality follows from $(w,j) \in E^{tight}(S)$. This yields a contradiction to Lemma 2.3.11 for the start time of $j \in U^B$. □

With these lemmas we have laid the foundation stone to a number of other results we will present in the following. We start with the earlier mentioned property of closed bulk-critical sets.

Lemma 2.3.13. *Let U^B a bulk-critical set for $(N = (V \cup W, E), p)$ and $\overline{U^B}$ its closure. Then $ImPred^{tight}(w) \cap U^B \neq \emptyset$ for every OR-node $w \in \overline{U^B}$.*

Proof. By contradiction suppose that there exist some OR-node $w \in \overline{U^B}$ such that $ImPred^{tight}(w) \cap U^B = \emptyset$. By Lemma 2.3.12 we know that there exists a $\delta > 0$ such that $S_w = S_w^{[V\setminus U^B+\varepsilon]}$ for all $0 < \varepsilon \leq \delta$. We fix this δ in the following. Note that by the choice of δ it holds that $E^{tight}(S^{[V\setminus U^B+\varepsilon]}) \subseteq E^{tight}(S)$. Consider the start times of w in S and $S^{[V\setminus U^B+\varepsilon]}$, it holds that

$$\begin{aligned} S_w^{[V\setminus U^B+\varepsilon]} &= \min\{S_v^{[V\setminus U^B+\varepsilon]} + p_v^{[V\setminus U^B+\varepsilon]} \mid (v,w) \in E^{tight}(S^{[V\setminus U^B+\varepsilon]})\} \\ &\geq \min\{S_v + p_v^{[V\setminus U^B+\varepsilon]} \mid (v,w) \in E^{tight}(S^{[V\setminus U^B+\varepsilon]})\} \\ &\geq \min\{S_v + p_v^{[V\setminus U^B+\varepsilon]} \mid (v,w) \in E^{tight}(S)\} \\ &= \min\{S_v + p_v + \varepsilon \mid (v,w) \in E^{tight}(S)\} \\ &= S_w + \varepsilon > S_w \end{aligned}$$

This yields a contradiction to Lemma 2.3.12 and proves the assertion. □

Now we will present a purely structural definition of two sets, which are equivalent to the path- and bulk-critical sets. Let an AND/OR-network $N = (V \cup W, E)$ and a processing time vector p be given. Consider the network N^{tight} induced by the tight edges in the earliest start schedule of (N,p) introduced in Section 1.2.2, and let $H(N^{tight}) = (V,A)$ be the hypergraph of the tight network N^{tight}, presented in Section 1.2.3.

Definition 2.3.14. *Let $H(N^{tight}) = (V,A)$ be the hypergraph of the tight network of $(N = (V \cup W, E), p)$. A set $U^\vee \subseteq V$ is called* OR-closed *in N, if it is inclusion-minimal with respect to the property*

(∨) $t \in U^\vee$ *and for every* $j \in U^\vee \setminus \{s\}$ $\exists\,(T,j) \in A : T \subseteq U^\vee$.

An OR-closed set $U^\vee$ of a network N is an inclusion-minimal subset of the jobs containing the sink t and having the following property. For every job j in the set $U^\vee$, there exists an incoming hyperedge (T,j) for which the whole tail T is also included in $U^\vee$. Thus, an OR-closed set is a hyperpath $\Pi_{\{s\}t}$ from root set $\{s\}$ to sink t as we have introduced it in Section 1.2.3. Translated to the notation of the AND/OR-network, this means that for every job $j \in U^\vee$ there exists a direct tight predecessor, which is either also in $U^\vee$ if it is an AND-node, or for which all direct tight predecessors are contained in $U^\vee$ if it is an OR-node. Similarly we can define AND-closed sets.

Definition 2.3.15. *Let* $H(N^{tight}) = (V,A)$ *be the hypergraph of the tight network of* $(N = (V \cup W, E), p)$. *A set* $U^\wedge \subseteq V$ *is called* AND-closed *in* N*, if it is inclusion-minimal with respect to the property*

(∧) $t \in U^\wedge$ *and for every* $j \in U^\wedge \setminus \{s\}$ $\forall\,(T,j) \in A : T \cap U^\wedge \neq \emptyset$.

In words, an AND-closed set $U^\wedge$ of a network N is an inclusion-minimal subset of the jobs containing sink t and having the following property. For every job j in $U^\wedge$, one member of the tail T of every incoming hyperedge (T,j) is also contained in $U^\wedge$. Again translated to the AND/OR-network notation, this means that for every job j in the set $U^\wedge$, for every incoming tight edge (v,j), either $v \in U^\wedge$ if v is a job, or one direct tight predecessor of v is in $U^\wedge$ if v is an OR-node.

Note that in both definitions, the sets $U^\vee$ and $U^\wedge$ have been defined for N but on the structure of the tight sub-network N^{tight} of N. Nevertheless, we talk about the OR- and AND-closed sets of N, bearing in mind that their structures are restricted to the tight sub-network.

We will prove that the OR- and AND-closed sets of a network (N,p) correspond to the path- and bulk-critical sets of (N,p). This gives us a nice structural characterisation of critical sets. First we have to consider the start time of the jobs in the sets again.

Lemma 2.3.16. *Let* $U^\vee$ *[*$U^\wedge$*] be an OR-closed [AND-closed] set of AND/OR-network* $(N = (V \cup W, E), p)$ *with earliest start time vector* S*. Then there exists an* $\varepsilon > 0$ *such that* $S_j^{[V \setminus U^\vee - \varepsilon]} = S_j$ *[*$S_j^{[V \setminus U^\wedge + \varepsilon]} = S_j$*] for all* $j \in U^\vee$ *[*$j \in U^\wedge$*],*

As in Remark 2.2.5 for the path- and bulk-critical sets, the existence of such an $\varepsilon > 0$ by the monotonicity of the earliest start time vector implies that $S_j^{[V \setminus U^\vee - \gamma]} = S_j$ and $S_j^{[V \setminus U^\wedge + \gamma]} = S_j$ for all $0 < \gamma \leq \varepsilon$ and for all $j \in U^\vee$ and $j \in U^\wedge$, respectively.

Proof. First we have to find an appropriate ε. Let

$$0 < \delta = \frac{slack(N,p)}{|V|+1}$$

and keep this δ fixed in the following. By this choice of δ we guarantee two things for every $0 < \varepsilon \leq \delta$. First, for any job processing time it holds that $p_j - \varepsilon > 0$. Second, $E^{tight}(S^\varepsilon) \subseteq E^{tight}(S)$ for $S^\varepsilon = S^{[V \setminus U^\vee - \varepsilon]}$ or $S^\varepsilon = S^{[V \setminus U^\wedge + \varepsilon]}$. During the whole proof we will always consider the tight edges with respect to S.

Now let $0 < \varepsilon \leq \delta$ and consider an OR-closed set $U^\vee$ of N with $S^\varepsilon = S^{[V \setminus U^\vee - \varepsilon]}$. By contradiction assume that there exists $j \in U^\vee$ such that $S_j^\varepsilon < S_j$. Among all jobs with this property take k with minimal start time S_k^ε.

By the definition of $U^\vee$ we know that for $k \in U^\vee$ there exists a hyperedge $a = (T_a, k)$ such that $T_a \subseteq U^\vee$. We have to consider the two cases that hyperedge a corresponds to a) an OR-node $w \in W$ or b) a standard edge $e \in E$ of the AND/OR-network N.
a) If $a = a(w) \in A^W$ then $ImPred^{tight}(w) = T_a \subseteq U^\vee$ and thus

$$\begin{aligned}\min\{S_i + p_i \mid (i,w) \in E^{tight}\} &= S_w = S_k > S_k^\varepsilon \geq S_w^\varepsilon \\ &= \min\{S_i^\varepsilon + p_i^\varepsilon \mid (i,w) \in E^{tight}\} \\ &= \min\{S_i^\varepsilon + p_i \mid (i,w) \in E^{tight}\}.\end{aligned}$$

The inequality implies that $S_i^\varepsilon < S_i$ for an $i \in ImPred^{tight}(w)$, in contradiction to the minimal choice of k.
b) If $a = (\{i\}, k) \in A^E$ we know that $i \in U^\vee$. Again consider the calculation of the start time of k.

$$\begin{aligned}S_k^\varepsilon &= \max\{S_j^\varepsilon + p_j^\varepsilon \mid (j,k) \in E\} \\ &\geq \max\{S_j^\varepsilon + p_j^\varepsilon \mid (j,k) \in E, j = i \in U^\vee\} \\ &= \max\{S_j + p_j \mid (j,k) \in E, j = i \in U^\vee\} = S_k\end{aligned}$$

The second inequality follows from the monotonicity of the maximum function, which can only decrease, if we decrease the number of elements considered. In the third equality, $p_i^\varepsilon = p_i$ as $i \in U^\vee$ and $S_i^\varepsilon = S_i$ as k was minimal with the property that $S_i^\varepsilon < S_i$ and i is a direct predecessor of k. The last equality holds by the definition of $U^\vee$. As $U^\vee$ is defined on $H(N^{tight})$, i is a tight direct predecessor of k and thus $S_k = S_i + p_i$. We get a contradiction to the assumption, which concludes the proof for OR-closed sets.

The case of AND-closed sets is slightly different to handle. To shorten notation let $S^\varepsilon = S^{[V \setminus U^\wedge + \varepsilon]}$. For the sake of contradiction, again assume that there exists $j \in U^\wedge$ such that $S_j^\varepsilon > S_j$. Among all jobs with this property take k with

minimal start time S_k. Consider the start times S_k and S_k^ε.

$$\begin{aligned} S_k &= \max\{S_v + p_v \mid (v,k) \in E\} \\ &= \max\{\, \max\{S_j + p_j \mid (j,k) \in E, j \in V\}, \\ &\qquad \max\{S_w + p_w \mid (w,k) \in E, w \in W\}\} \end{aligned}$$

$$\begin{aligned} S_k^\varepsilon &= \max\{S_v^\varepsilon + p_v^\varepsilon \mid (v,k) \in E\} \\ &= \max\{\, \max\{S_j^\varepsilon + p_j^\varepsilon \mid (j,k) \in E, j \in V\}, \\ &\qquad \max\{S_w^\varepsilon + p_w^\varepsilon \mid (w,k) \in E, w \in W\}\}. \end{aligned}$$

We will treat the two cases of predecessors of k individually.
a) Consider $j \in \mathit{ImPred}^{tight}(k) \cap V$. By definition of $U^\wedge$, it follows that $j \in U^\wedge$ thus $p_j^\varepsilon = p_j$. By the minimal choice of k we also know that $S_j^\varepsilon = S_j$. Together this implies

$$\max\{S_j + p_j \mid (j,k) \in E^{tight}, j \in V\} = \max\{S_j^\varepsilon + p_j^\varepsilon \mid (j,k) \in E^{tight}, j \in V\}$$

b) Now consider $w \in \mathit{ImPred}^{tight}(k) \cap W$. Of course we have $p_w^\varepsilon = p_w = 0$. By Definition 2.3.15 we know that for every $w \in \mathit{ImPred}^{tight}(k) \cap W$, there exists $x \in \mathit{ImPred}^{tight}(w)$ such that $x \in U^\wedge$. We can establish the following inequality for the start time of each OR-node $w \in \mathit{ImPred}^{tight}(k) \cap W$.

$$\begin{aligned} S_w^\varepsilon &= \min\{S_j^\varepsilon + p_j^\varepsilon \mid j \in \mathit{ImPred}^{tight}(w)\} \\ &\leq \min\{S_j^\varepsilon + p_j^\varepsilon \mid j \in \mathit{ImPred}^{tight}(w), j \in U^\wedge\} \\ &= \min\{S_j + p_j \mid j \in \mathit{ImPred}^{tight}(w), j \in U^\wedge\} \\ &= \min\{S_j + p_j \mid j \in \mathit{ImPred}^{tight}(w)\} = S_w \end{aligned}$$

The second inequality is true by the monotonicity of the minimum function. The third equality holds by definition of $U^\wedge$ and the minimal choice of k. The fourth equality again holds by definition of $U^\wedge$ as it contains one tight direct predecessor of the OR-node, thus preserving the minimum. As this inequality holds for all OR-nodes $w \in \mathit{ImPred}^{tight}(k) \cap W$, we get that

$$\max\{S_w \mid (w,k) \in E^{tight}, w \in W\} \geq \max\{S_w^\varepsilon \mid (w,k) \in E^{tight}, w \in W\}$$

Altogether we have a contradiction to our assumption that $S_j^\varepsilon > S_j$ which completes the proof for the AND-closed sets as well. □

Now we are able to prove the equivalence between path-critical and OR-closed sets and between bulk-critical and AND-closed sets.

Theorem 2.3.17. *For an AND/OR-network $(N = (V \cup W, E), p)$, a set $U \subseteq V$ is path-critical if and only if it is OR-closed.*

Proof. It is easy to see that an OR-closed set $U^\vee$ has property (P). By definition, $t \in U^\vee$ and with Lemma 2.3.16 we get there exists $\varepsilon > 0$ such that $C_{\max}(S^{[V\setminus U^\vee-\varepsilon]}) = S_t^{[V\setminus U^\vee-\varepsilon]} = S_t = C_{\max}(S)$.

Now consider a path-critical set U^P and suppose it does not have property ($\vee$). This means that there exists a job $j \in U^P$ such that for all incoming hyperedges $a = (T_a, j)$, $T_a \not\subseteq U^P$. We want to compare the two start times of j in schedule S and $S^\varepsilon = S^{[V\setminus U^\vee-\varepsilon]}$ for some small ε fulfilling Lemma 2.3.16. Remember that $\varepsilon > 0$ is small enough to guarantee that $E^{tight}(S^\varepsilon) \subseteq E^{tight}(S)$. Consider S_j and S_j^ε.

$$\begin{aligned}
S_j &= \max\{S_v + p_v \mid (v,j) \in E\} \\
&= \max\{S_v + p_v \mid (v,j) \in E^{tight}(S)\} \\
&= \max\{\max\{S_i + p_i \mid (i,j) \in E^{tight}(S), i \in V\}, \\
&\qquad \max\{S_w \mid (w,j) \in E^{tight}(S), w \in W\}\}
\end{aligned}$$

$$\begin{aligned}
S_j^\varepsilon &= \max\{S_v^\varepsilon + p_v^\varepsilon \mid (v,j) \in E\} \\
&= \max\{S_v^\varepsilon + p_v^\varepsilon \mid (v,j) \in E^{tight}(S^\varepsilon)\} \\
&= \max\{\max\{S_i^\varepsilon + p_i^\varepsilon \mid (i,j) \in E^{tight}(S^\varepsilon), i \in V\}, \\
&\qquad \max\{S_w^\varepsilon \mid (w,j) \in E^{tight}(S^\varepsilon), w \in W\}\}.
\end{aligned}$$

We have to distinguish between two cases, a) a corresponds to an edge or b) a corresponds to an OR-node.
a) $a = a(e) = (\{i\}, j) \in A^E$ and $i \in V$. Then by assumption we know that $i \notin U^P$ and therefore

$$\begin{aligned}
&\max\{S_i^\varepsilon + p_i^\varepsilon \mid (i,j) \in E^{tight}(S^\varepsilon), i \in V\} \\
=&\max\{S_i^\varepsilon + p_i - \varepsilon \mid (i,j) \in E^{tight}(S^\varepsilon), i \in V\} \\
\leq&\max\{S_i + p_i - \varepsilon \mid (i,j) \in E^{tight}(S^\varepsilon), i \in V\} \\
\leq&\max\{S_i + p_i - \varepsilon \mid (i,j) \in E^{tight}(S), i \in V\} \\
=&\max\{S_i + p_i \mid (i,j) \in E^{tight}(S), i \in V\} - \varepsilon
\end{aligned}$$

b) For $a = a(w) \in A^W$ we can do a similar calculation. For every OR-node w with $(w,j) \in E^{tight}(S)$ it holds that

$$\begin{aligned}
S_w^\varepsilon &= \min\{S_i^\varepsilon + p_i^\varepsilon \mid (i,w) \in E\} \\
&\leq \min\{S_i + p_i^\varepsilon \mid (i,w) \in E\} \\
&\leq \min\{S_i + p_i^\varepsilon \mid (i,w) \in E^{tight}(S)\} \\
&\leq \min\{S_i + p_i^\varepsilon \mid (i,w) \in E^{tight}(S), i \notin U^P\} \\
&= \min\{S_i + p_i - \varepsilon \mid (i,w) \in E^{tight}(S), i \notin U^P\} = S_w - \varepsilon
\end{aligned}$$

The last equality holds as, by assumption, there exist a direct tight predecessor i of w that is not in U^P and thus $p_i^\varepsilon = p_i - \varepsilon$. So the minimum will be attained in this predecessor. Bringing the cases a) and b) together yields that $S_j^\varepsilon \leq S_j - \varepsilon$ which is a contradiction to Lemma 2.3.11.

It is left to show the inclusion minimality of the sets with the required property. This follows automatically from the inclusion-minimality of the sets with their properties. Suppose for example that U^P is not inclusion-minimal with respect to property ($\vee$). Then there exists a proper subset $U \subsetneq U^P$ that is inclusion-minimal with respect to property ($\vee$). But then, U is OR-closed and thus has property (P), as we have just shown. This is a contradiction to the inclusion-minimality of U^P with respect to property (P). Exactly the same argumentation yields the inclusion-minimality of $U^\vee$ with respect to property (P), concluding the proof. □

Theorem 2.3.18. *For an AND/OR-network $(N = (V \cup W, E), p)$, a set $U \subseteq V$ is bulk-critical if and only if it is AND-closed.*

Proof. It is easy to see that an AND-closed set $U^\wedge$ has property (B). By definition, $t \in U^\wedge$ and with Lemma 2.3.16 we get that there exists an $\varepsilon > 0$ such that $C_{\max}(S^{[V\setminus U^\wedge+\varepsilon]}) = S_t^{[V\setminus U^\wedge+\varepsilon]} = S_t = C_{\max}(S)$.

Now consider a bulk-critical set U^B and suppose it does not have property ($\wedge$). This means that there exists a job $j \in U^B$ such that there exists an incoming hyperedge $a = (T_a, j)$ with $T_a \cap U^B = \emptyset$. We want to have a look at the two start times S_j and $S_j^\varepsilon = S_j^{[V\setminus U^\wedge+\varepsilon]}$ of j, where ε is chosen such that it fulfills Lemma 2.3.16 and small enough to guarantee $E^{tight}(S^\varepsilon) \subseteq E^{tight}(S)$. As in the proof of Theorem 2.3.17, we distinguish between AND- and OR-predecessors in the calculation of the start time of j.

$$\begin{aligned} S_j = \max\{ &\max\{S_i + p_i \mid (i,j) \in E^{tight}(S), i \in V\}, \\ &\max\{S_w \mid (w,j) \in E^{tight}(S), w \in W\}\} \end{aligned}$$

$$\begin{aligned} S_j^\varepsilon = \max\{ &\max\{S_i^\varepsilon + p_i^\varepsilon \mid (i,j) \in E^{tight}(S^\varepsilon), i \in V\}, \\ &\max\{S_w^\varepsilon \mid (w,j) \in E^{tight}(S^\varepsilon), w \in W\}\}. \end{aligned}$$

Accordingly we consider the two cases that a corresponds to a) an edge or b) an OR-node.

a) $a = a(e) = (\{i\}, j) \in A^E$ and $i \in V$. Then $i \notin U^B$ and therefore

$$\begin{aligned} &\max\{S_i^\varepsilon + p_i^\varepsilon \mid (i,j) \in E, i \in V\} \\ \geq &\max\{S_i + p_i^\varepsilon \mid (i,j) \in E, i \in V\} \\ \geq &\max\{S_i + p_i^\varepsilon \mid (i,j) \in E^{tight}(S), i \in V\} \\ = &\max\{S_i + p_i + \varepsilon \mid (i,j) \in E^{tight}(S), i \in V\} \\ = &\max\{S_i + p_i \mid (i,j) \in E^{tight}(S), i \in V\} + \varepsilon \end{aligned}$$

b) For $a = a(w) \in A^W$ we can do a similar calculation. For the start time of every OR-node w with $ImPred^{tight}(w) \cap U^B = \emptyset$, where $ImPred^{tight}(w)$ are the tight direct predecessors of w relative to S, we get

$$\begin{aligned} S^\varepsilon_w &= \min\{S^\varepsilon_i + p^\varepsilon_i \mid (i,w) \in E^{tight}(S^\varepsilon)\} \\ &\geq \min\{S_i + p^\varepsilon_i \mid (i,w) \in E^{tight}(S^\varepsilon)\} \\ &\geq \min\{S_i + p^\varepsilon_i \mid (i,w) \in E^{tight}(S)\} \\ &= \min\{S_i + p_i + \varepsilon \mid (i,w) \in E^{tight}(S)\} = S_w + \varepsilon \end{aligned}$$

Combining cases a) and b) we get $S^\varepsilon_j \geq S_j + \varepsilon$, a contradiction to Lemma 2.3.11.

The inclusion-minimality of the sets with respect to the required property follows from the inclusion-minimality of the sets with respect to their defining property by the standard argument. We have stated this argument before at the end of the proof of Theorem 2.3.17, for example. □

We have given some quantitative and some structural characterisations of the different critical sets in this section. All of them will be useful for further consideration. We will change the subject a little bit and treat complexity questions.

2.4 The Complexity of Finding Critical Jobs and Sets

In this section we will discuss different natural questions about the complexity of criticality. Is it easy to find a critical set? How difficult is it to decide whether a job is critical in one of the four senses? It will turn out that there is a fundamental complexity gap between these two questions, which do not look that different on the first sight.

At the end of the section we make a short excursion into logic. We will see that a complexity result for critical jobs carries over to a certain satisfiability question on monotone Boolean functions.

2.4.1 The Complexity of Constructing Critical Sets

The first question we want to answer is, how difficult it is to find a critical set. The answer is simple, it is easy, as we prove in the next theorem.

Theorem 2.4.1. *For an AND/OR-network $(N = (V \cup W, E), p)$, a cut-, path-, delay-, or bulk-critical set $U \subseteq V$ can be obtained in time polynomial in $size(N)$.*

To ease the proof of the theorem we will show a lemma first. This lemma tells us that for the right choice of a small constant ε we get the following property for ε: If changing the processing times of a set of jobs by ε changes the completion

time of some job j, then a change of the processing times of the same set of jobs by some smaller constant than ε also changes the completion time of j.

Lemma 2.4.2. *Let $(N = (V \cup W, E), p)$ be an AND/OR-network, $U \subseteq V$ a set of jobs and $0 < \varepsilon < slack(N,p)/|U|$ some small constant. Then $C_v^{[U-\varepsilon']} < C_v$ if and only if $C_v^{[U-\varepsilon]} < C_v$ and $C_v^{[U+\varepsilon']} > C_v$ if and only if $C_v^{[U+\varepsilon]} > C_v$ for any $0 < \varepsilon' < \varepsilon$ and for every node $v \in V \cup W$.*

Proof. Let $0 < \varepsilon' < \varepsilon < slack(N,p)/|U|$, then by Lemma 2.2.7 and Corollary 2.2.8, $E^{tight}(S^{[U-\varepsilon]}) \subseteq E^{tight}(S^{[U-\varepsilon']}) \subseteq E^{tight}(S)$. By the monotonicity of the completion times we get that if $C_v^{[U-\varepsilon']} < C_v$ then also $C_v^{[U-\varepsilon]} < C_v$ and if $C_v^{[U+\varepsilon']} > C_v$ then also $C_v^{[U+\varepsilon]} > C_v$.

For the other implication consider the case of a decrease in processing times and suppose that there exists a node $v \in V \cup W$ with $C_v^{[U-\varepsilon]} < C_v^{[U-\varepsilon']} = C_v$. From $C_v^{[U-\varepsilon']} = C_v$ follows that $v \notin U$ and $p_v^{[U-\varepsilon]} = p_v^{[U-\varepsilon']} = p_v$. Let $v \in V \cup W$ with $C_v^{[U-\varepsilon]} < C_v^{[U-\varepsilon']} = C_v$ be the node with minimal completion time $C_v^{[U-\varepsilon]}$ among all nodes with this property. If v is not unique, we prefer tight predecessors in the case that they are related and otherwise we take an arbitrary one. Due to this choice, v cannot be an OR-node, as the tight predecessor of an OR-node has the same completion time as the OR-node and would be preferred. As $S^{[U-\varepsilon]}$ is an earliest start schedule, there exist a tight direct predecessor u of v in $S^{[U-\varepsilon]}$, that is $(u,v) \in E^{tight}(S^{[U-\varepsilon]})$. From $E^{tight}(S^{[U-\varepsilon]}) \subseteq E^{tight}(S^{[U-\varepsilon']}) \subseteq E^{tight}(S)$ and the minimal choice of v we establish the following contradiction:

$$C_v = C_u + p_v > C_u^{[U-\varepsilon']} + p_v = C_v^{[U-\varepsilon']} = C_v.$$

Now suppose that there exists a node $v \in V \cup W$ with $C_v^{[U+\varepsilon]} > C_v^{[U+\varepsilon']} = C_v$. We follow the same argumentation and conclude that $v \notin U$ which implies that $p_v^{[U+\varepsilon]} = p_v^{[U+\varepsilon']} = p_v$. Again consider v with minimal completion time $C_v^{[U+\varepsilon]}$. If there are several such nodes, we prefer tight predecessors if the nodes are related, and we take an arbitrary one otherwise. Due to this choice, v cannot be an OR-node. Consider a tight direct predecessor u of v with $(u,v) \in E^{tight}(S^{[U+\varepsilon]})$. Node u is also a tight direct predecessor of v with respect to the other schedules $S^{[U+\varepsilon']}$ and S. By the minimal choice of v we get that

$$C_v = C_u + p_v < C_u^{[U+\varepsilon']} + p_v = C_v^{[U+\varepsilon']} = C_v,$$

which is a contradiction and completes the proof. □

Proof of Theorem 2.4.1. Let us show how to find a cut-critical set $U^C \subseteq V$, for example. We start with the set of all jobs in V and take out jobs one by one until we are left with a cut-critical set at the end. Obviously, $U = V$ has property (C)

but most likely it will not be inclusion-minimal with respect to property (C). We need an appropriate ε to check property (C) for some chosen set. Therefore let

$$0 < \varepsilon < \frac{slack(N,p)}{|V|}.$$

Suppose $C_{\max}(S^{[U-\varepsilon]}) < C_{\max}(S)$, then by Lemma 2.4.2, we get $C_{\max}(S^{[U-\varepsilon']}) < C_{\max}(S)$ for every $0 < \varepsilon' < \varepsilon$ and by the monotonicity of the makespan stated in Lemma 1.2.5 we also know that $C_{\max}(S^{[U-\varepsilon'']}) < C_{\max}(S)$ for every $\varepsilon'' > \varepsilon$. Together this yields that if for a set U, $C_{\max}(S^{[U-\varepsilon]}) < C_{\max}(S)$, then set U fulfills property (C). Therefore property (C) can be checked by one earliest start calculation in polynomial time. The same argumentation holds for property (D).

To be able to find a path- or bulk-critical set we also have to argue why checking a set U with ε suffices to decide whether U fulfills property (P) or (B), respectively. Consider a set U with $C_{\max}(S^{[V\setminus U-\varepsilon]}) < C_{\max}(S)$. We have just shown that this implies $C_{\max}(S^{[V\setminus U-\varepsilon']}) < C_{\max}(S)$ for all $\varepsilon' > 0$ and thus set U cannot fulfill property (P). The same argumentation holds for property (B). So again we can check whether a set has the required property or not by one earliest start calculation.

It remains to show that we only have to check a polynomial number of sets. Consider the cut-critical case and a set U with property (C). Take an arbitrary $j \in U$ and check whether $U \setminus \{j\}$ still has property (C) by computing $S^{[U\setminus\{j\}-\varepsilon]}$. If $U \setminus \{j\}$ has property (C) we take j out of U and continue. If $U \setminus \{j\}$ does not have property (C) we leave j in U. By the monotonicity of the makespan we know that if $U \setminus \{j\}$ does not have property (C) no proper subset $U' \subsetneq U \setminus \{j\}$ will have property (C). So we do not have to consider j again. When we have checked every vertex $j \in V$ once, we are left with a cut-critical set $U^C = U$. Obviously the procedure needs $|V|$ calls of the polynomial time Algorithm 2, which is again polynomial in $size(N)$. For this argument we only needed the monotonicity of the makespan and thus it also holds for the other three cases, which completes the proof.

□

2.4.2 The Complexity of Identifying Critical Jobs

In the last section we have seen that finding some critical set is easy. A natural question to ask is, whether it is also easy to find a critical set containing a specific job. This is equivalent to the question whether it is easy to decide if some job is critical, which is a simple task in a standard project network. The next theorem will give a negative answer to this question.

Theorem 2.4.3. *For an AND/OR-network $(N = (V \cup W, E), p)$ it is NP-complete to decide whether a given job j is cut- (path-, delay-, or bulk-) critical.*

To prove that it is difficult to find a cut-critical or delay-critical set containing a specific job j we give a reduction from SAT.

> SAT: Let X be a set of Boolean variables and Ψ a collection of clauses. Does there exist a satisfying truth assignment for Ψ?

Recall that SAT is one of the basic NP-complete problems stated in Garey and Johnson (1979).

Proof. Let us again focus on the cut-critical case first. Given a specific job $j \in V$, we want to decide whether j is cut-critical or not. A cut-critical set U^C containing job j is a certificate to check in polynomial time that a job j is cut-critical, so the problem is in NP.

Consider an instance Ψ of SAT with k clauses $C_1, \ldots, C_k$ on ℓ variables $X := \{x_1, \ldots, x_\ell\}$. For each variable x_λ we introduce three AND-nodes, labeled by x_λ itself, its negation $\overline{x}_\lambda$, and y_λ, $\lambda = 1, \ldots, \ell$. For each clause we introduce an OR-node C_κ, $\kappa = 1, \ldots, k$. In addition we need an AND-node for job j, a dummy OR-node w, and a dummy AND-node z. The completion of the project is represented by the AND-node t. All AND-nodes have processing time $p = 1$, the OR-nodes have processing time $p = 0$, as usual.

The arc set E can be divided into two disjoint sets E_1 and E_2. For each $\lambda = 1, \ldots, \ell$ there are the three arcs (x_λ, y_λ), $(\overline{x}_\lambda, y_\lambda)$, and (y_λ, w). Those arcs belong to E_1. For each clause C_κ, $\kappa = 1, \ldots, k$, there are $|C_\kappa|$ arcs going into the clause and one outgoing arc (C_κ, z). The in-going arcs start at the nodes corresponding to the variables used in the clause, taking into account the correct sign. That is, if clause C contains variable x_q in positive form and x_r in negated form, then there are the arcs (x_q, C) and $(\overline{x}_r, C)$ in the network. Those arcs together with (j, z), (z, w) and (w, t) compose arc set E_2. In Figure 2.6 the network corresponding to instance

$$\Psi := \underbrace{(x_1 \vee x_2)}_{C_1} \wedge \underbrace{(\overline{x}_2 \vee \overline{x}_3)}_{C_2} \wedge \underbrace{(\overline{x}_1 \vee \overline{x}_2 \vee x_3)}_{C_3} \wedge \underbrace{(\overline{x}_1 \vee x_2 \vee \overline{x}_3)}_{C_4}$$

of SAT is given. The dashed arcs belong to E_1, while the others are contained in E_2.

Claim: There exists a fulfilling assignment for Ψ if and only if j is cut-critical.

Let $f : X \to \{\text{TRUE}, \text{FALSE}\}$ be a fulfilling assignment for instance Ψ. Let U be the set containing j and for all $\lambda = 1, \ldots, \ell$, node x_λ if $f(x_\lambda) = \text{TRUE}$, otherwise node $\overline{x}_\lambda$. Due to the construction of the Network, U has property (C):

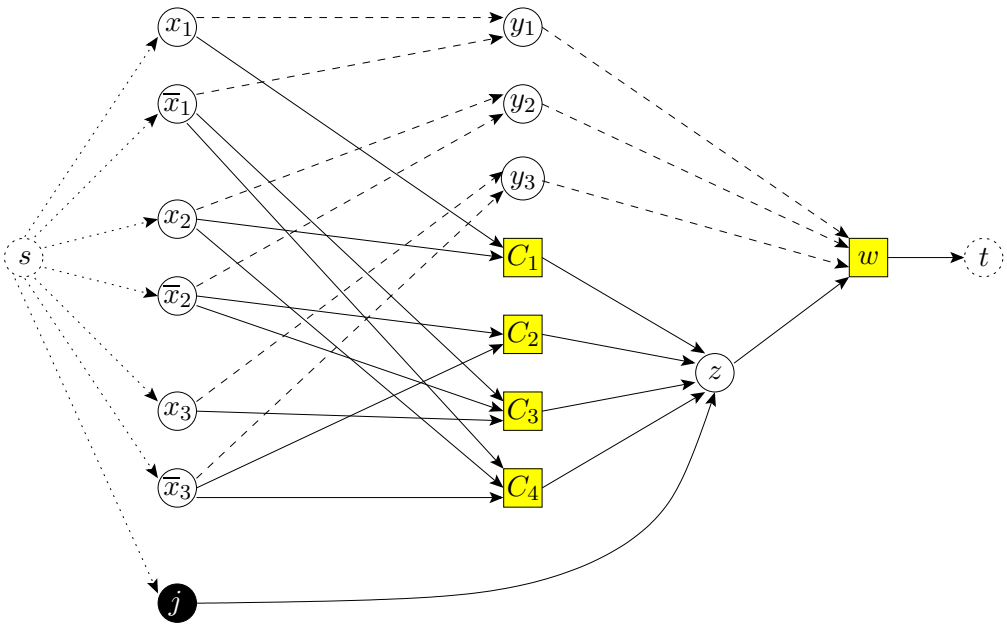


Figure 2.6: AND/OR-network N^C representing instance Ψ of SAT such that a cut-critical set U^C containing j corresponds to a fulfilling assignment for Ψ.

every clause is fulfilled by a truth assignment thus one of its literals is true and contained in U and the OR-node of the clause can start earlier. The start time of the dummy AND-node z gets reduced through the decrease of j's processing time and the earlier start of the clause nodes, thus reducing the start time of the dummy OR-node w. But U might be not inclusion-minimal with respect to this property. Let U^C be an inclusion-minimal subset of U with respect to (C). It holds that $j \in U^C$, as in a fulfilling assignment, each variable is either TRUE or FALSE, so for any $\lambda = 1, \ldots, \ell$, only x_λ or $\overline{x}_\lambda$ can be in U^C, but not both. Hence none of the y_λ can start earlier and the only possibility to reduce the makespan is to decrease the start time of dummy AND-node z. As j is an immediate predecessor of z, node j is contained in any inclusion-minimal subset of U with respect to (C).

Conversely let j be cut-critical with set U^C. The set U^C cannot contain x_λ and $\overline{x}_\lambda$ for any $\lambda = 1, \ldots, \ell$, as it is $\subseteq$-minimal with respect to property (C) and contains j. If $x_\lambda \in U^C$ and $\overline{x}_\lambda \in U^C$ for some $\lambda \in \{1, \ldots, \ell\}$, then the start time of y_λ would be decreased, thus enabling dummy OR-node w to start earlier and making j in U^C redundant. Define $f : X \to \{\text{TRUE}, \text{FALSE}\}$ as follows:

$$f(x_\lambda) := \begin{cases} \text{TRUE} & \text{if } x_\lambda \in U^C, \\ \text{FALSE} & \text{otherwise.} \end{cases}$$

Obviously f is a fulfilling assignment for instance Ψ as every OR-node repre-

senting a clause has to start earlier to decrease the start time of z. This can only be achieved if for every clause one literal is contained in U^C, thus being true in our assignment. This completes the proof for the cut-critical and by Theorem 2.3.3 as well for the path-critical case.

Now we consider the delay-critical case. Given a specific job $j \in V$, we want to decide whether j is delay-critical or not. Again, a delay-critical set U^D containing job j is a certificate to check in polynomial time that a job j is delay-critical, so the problem is in NP.

To prove that it is also difficult to find such a certificate U^D for j we give a reduction from SAT to the problem *"j delay-critical?"*. Consider an instance Ψ of SAT with k clauses $C_1, \ldots, C_k$ on ℓ variables $X := \{x_1, \ldots, x_\ell\}$. For each variable x_λ, $\lambda = 1, \ldots, \ell$, we introduce six AND-nodes, x_λ itself, its negation $\overline{x}_\lambda$, z_λ, $\overline{z}_\lambda$, y_λ, and i_λ. We assign the following processing times to those nodes:

- $p(x_\lambda) = p(\overline{x}_\lambda) = \lambda$, and
  $p(z_\lambda) = p(\overline{z}_\lambda) = \ell - \lambda + 1$, for all $\lambda = 1, \ldots, \ell$,
- $p(y_\lambda) = p(i_\lambda) = 1$, for $\lambda = 1, \ldots, \ell - 1$
  $p(y_\ell) = p(i_\ell) = 2$.

In addition we introduce three OR-nodes w_λ, $\overline{w}_\lambda$, and u_λ for each $\lambda = 1, \ldots, \ell$. For each clause we introduce an OR-node C_κ, $\kappa = 1, \ldots, k$. We also need an AND-node for job j and a dummy AND-node v, with processing times $p(j) = p(v) = 1$. The completion of the project is represented by the AND-node t. The OR-nodes have processing time $p = 0$, as usual.

The arc set E can be divided into three disjoint sets E_{solid}, E_{dot}, and E_{dash} and includes the following edges.

- E_{solid} contains six edges for each $\lambda = 1, \ldots, \ell$. These are (x_λ, w_λ), $(\overline{x}_\lambda, \overline{w}_\lambda)$, (w_λ, i_λ), and $(\overline{w}_\lambda, i_\lambda)$ for all $\lambda = 1, \ldots, \ell$. Additionally (j, w_1), $(j, \overline{w}_1)$ for $\lambda = 1$ and $(i_{\lambda-1}, w_\lambda)$, $(i_{\lambda-1}, \overline{w}_\lambda)$ for the other $\lambda = 2, \ldots, \ell$. We also count the edge (i_ℓ, t) into E_{solid}.
- E_{dot} contains four edges for every $\lambda = 1, \ldots, \ell$, namely (x_λ, u_λ), $(\overline{x}_\lambda, u_\lambda)$, (u_λ, y_λ), and except for $\lambda = \ell$ the edge $(y_\lambda, y_{\lambda+1})$. Instead, the last edge (y_ℓ, t) is also included into E_{dot}.
- E_{dash} is responsible for the clauses. First, for every $\lambda = 1, \ldots, \ell$, we introduce the two edges (x_λ, z_λ) and $(\overline{x}_\lambda, \overline{z}_\lambda)$ into E_{dash}. For every clause C_κ, $\kappa = 1, \ldots, k$, we include all edges from the literals that appear in the clause to the OR-node C_κ. The literals are represented by the z-nodes. Thus if clause $C_i = (x_q \vee \overline{x}_r)$, for example, then there are the edges (z_q, C_i) and $(\overline{z}_r, C_i)$ in E_{dash}. Additionally, for every $\kappa = 1, \ldots, k$, we introduce an edge (C_κ, v) and assume (v, t) to be in E_{dash} as well.

In Figure 2.7 the network N^D corresponding to the same instance $\Psi = (x_1 \vee x_2) \wedge (\overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3)$ as above is given. For the sake of clarity, we have not included the start vertex s and the edges from s to j and every x_λ and $\overline{x}_\lambda$. Instead, we have written the processing times of the jobs next to the nodes, as they are not unit in this case.

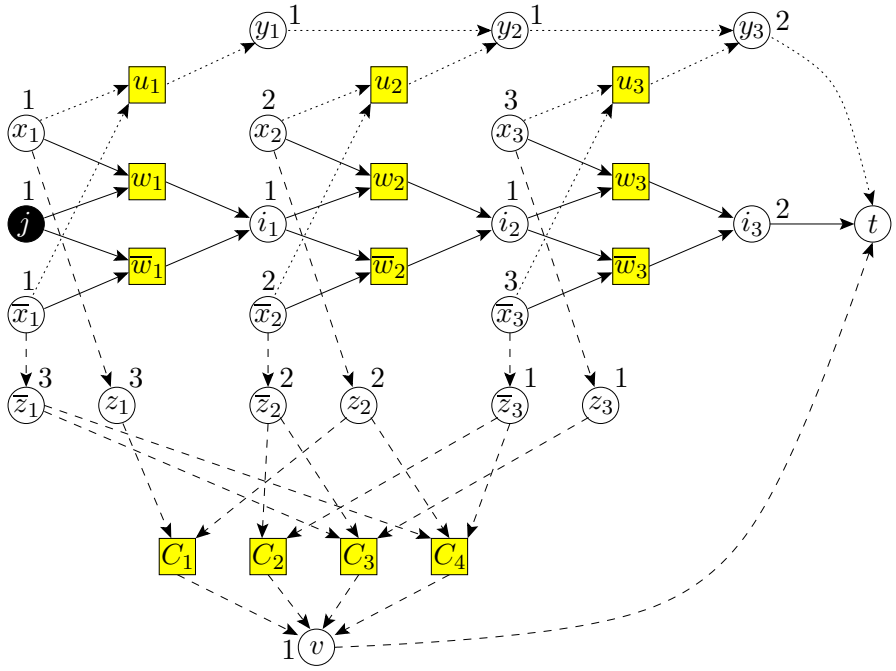


Figure 2.7: AND/OR-network N^D representing instance Ψ of SAT such that a delay-critical set U^D containing j corresponds to a fulfilling assignment for Ψ.

Claim: There exists a fulfilling assignment for Ψ if and only if j is delay-critical.

In contrast to the cut-critical case above, we identify the change in processing time of a job with the value FALSE of a variable. Therefore if $f : X \to \{\text{TRUE}, \text{FALSE}\}$ is some assignment for the variables, let U^D be the set containing j and for all $\lambda = 1, \ldots, \ell$, node x_λ if $f(x_\lambda) = \text{FALSE}$, otherwise node $\overline{x}_\lambda$.

First consider the sub-network N_{solid} induced by the solid edges in E_{solid}. This sub-network N_{solid} is built up such that we need at least one node out of $\{x_\lambda, \overline{x}_\lambda\}$ for every $\lambda = 1, \ldots, \ell$ to make j delay-critical. If the processing time of j is increased, then this delay can be passed through to t by either going via w_λ, then we need also to increase $p(x_\lambda)$ or via $\overline{w}_\lambda$, where we need $\overline{x}_\lambda$ for j to be in U^D. In a fulfilling assignment, a variable x_λ is either TRUE or FALSE and thus $\overline{x}_\lambda \in U^D$, if variable x_λ is true, and $x_\lambda \in U^D$ otherwise.

Now consider sub-network N_{dot} induced by E_{dot}. Network N_{dot} ensures that j is only delay-critical if at most one node out of $\{x_\lambda, \overline{x}_\lambda\}$ is in U^D for every

$\lambda = 1, \ldots, \ell$. Suppose that this is not the case and $x_\lambda \in U^D$ and $\overline{x}_\lambda \in U^D$ for some $1 \leq \lambda \leq \ell$. Then the increase of the processing times of x_λ and $\overline{x}_\lambda$ delays the start time of OR-node u_λ and thus also the start time of its successor y_λ. Every y_λ is a tight predecessor of t, connected by a simple tight directed path only containing AND-nodes. Thus the delay of the start time of any y_λ directly increases the makespan and thereby makes j redundant in the delay-critical set U^D. If j is critical with some set U^D, then by the construction of N_{dot}, U^D can contain at most one node out of $\{x_\lambda, \overline{x}_\lambda\}$.

Together, the two sub-networks N_{solid} and N_{dot} ensure that j with set U^D can be delay-critical if and only if f is a valid assignment.

It is left to consider the network N_{dash} induced by the set E_{dash} of dashed edges. This sub-network takes care about the fulfilling of the clauses. Suppose now that f is actually a satisfying truth assignment for Ψ. Then we know that for each clause C_κ, $\kappa = 1, \ldots, \ell$, at least one of the literals in C_κ is true and thus the corresponding x-node is not in U^D. None of the z-nodes is in U^D either, so the start time of C_κ is preserved for every $\kappa = 1, \ldots, \ell$. This yields that v does not get delayed. As a fulfilling truth assignment is of course valid, the two networks N_{solid} and N_{dot} then assure that U^D is delay-critical. Conversely let j be delay-critical with set U^D and define $f : X \to \{\text{TRUE}, \text{FALSE}\}$ as follows:

$$f(x_\lambda) := \begin{cases} \text{FALSE} & \text{if } x_\lambda \in U^D, \\ \text{TRUE} & \text{otherwise.} \end{cases}$$

Obviously, by the construction of the solid and the dotted sub-networks, this yields a valid assignment. Suppose now, that it is no satisfying truth assignment. Then there exists a clause C_κ that is not fulfilled, which can only be the case if every literal in it is false. If a literal is false, then the x-vertex representing the variable in the corresponding form is in U^D by the definition of the assignment. If we increase the processing times of the nodes in U^D, the start times of the corresponding z-nodes get delayed and thus the start time of OR-node C_κ gets delayed, as there is an edge from the z-nodes corresponding to the literals in C_κ. If one of the OR-nodes C_κ, $\kappa = 1, \ldots, \ell$ starts later, then also v has to start later, which increases the makespan. Thus if f does not fulfill one of the clauses, U^D contains a set of literals delaying these clauses, again making j redundant in U^D. This completes the proof for the delay-critical case and Corollary 2.3.6 extends the statement to the bulk-critical case as well. □

We have proved that, in contrast to the case of a standard network, deciding whether a specific job is critical in any of the four senses is NP-complete. A direct result of this theorem is that finding the set of all critical jobs is also NP-complete, again in contrast to the classical case.

Corollary 2.4.4. *For an AND/OR-network $(N = (V \cup W, E), p)$ it is NP-complete to identify the set of all cut-, path-, delay-, or bulk-critical jobs.*

In the next section we want to use this complexity result to prove a complexity result for monotone Boolean functions.

2.4.3 Criticality and Monotone Boolean Functions

Inspired by the structural characterisation of path- and bulk-critical sets presented in Section 2.3.4 we want to investigate critical sets from a logic point of view in this section. Möhring, Skutella, and Stork (2000b) establish a close relation between the feasibility of AND/OR-networks and HORN-SAT. An instance of SAT is of *Horn type*, if each clause contains at most one positive literal. Möhring et al. (2000b) show that a (bipartite) AND/OR-network induces an instance of HORN-SAT such that this instance of HORN-SAT is feasible if and only if the AND/OR-network contains a generalised cycle. We will relate the complexity of finding a critical set in an AND/OR-network to the complexity of fulfilling a monotone Boolean function. This will constitute a complexity result for satisfying monotone Boolean functions, which is new to our knowledge.

We will assume the reader to be familiar with the very basic notations of logic and referred to Prestel (1992) or Wegener (1987) for an extended introduction. Define TRUE to be equal to the value 1 and FALSE to be equal to 0. Then a *monotone Boolean function* in the variable $x = (x_1, \ldots, x_n)$ is a Boolean function $f(x)$ with the property that it is monotone increasing. A function $f(x)$ is called *monotone increasing* if for any two variables x, y with $x_i \leq y_i$ for all $i = 1, \ldots, n$ it follows that $f(x) \leq f(y)$.

Consider an AND/OR-network $N = (V \cup W, E)$, with processing time vector p. Let N^{tight} be its tight sub-network and $H(N^{tight})$ the hypergraph of N^{tight}. We will present several Boolean functions based on the structure of $H(N^{tight})$, which correspond to critical sets, as we will see. Each Boolean function will be based on the variable set $x = (x_s, x_1, \ldots, x_n, x_t)$, where variable x_j corresponds to job $j \in V$. The only Boolean operators used will be $\vee$ and $\wedge$, representing the "OR" and the "AND".

Definition 2.4.5. *For an AND/OR-network (N, p) we define the following four monotone Boolean functions based on the hypergraph $H(N^{tight})$.*

- $f^C(x) = f_t^C(x)$ *is called* cut-function *if*

$$f_j^C(x) := \begin{cases} \bigwedge_{(T,j)\in A} \left(\bigvee_{i \in T} (x_i \vee f_i^C(x)) \right) & \text{for } j \notin ImSucc^{tight}(s) \\ \text{FALSE} & \text{for } j \in ImSucc^{tight}(s) \end{cases}$$

- $f^P(x) = x_t \wedge f_t^P(x)$ *is called* path-function *if*

$$f_j^P(x) := \begin{cases} \bigvee_{(T,j)\in A} \left(\bigwedge_{i\in T}(x_i \wedge f_i^P(x))\right) & \text{for } j \notin ImSucc^{tight}(s) \\ \text{TRUE} & \text{for } j \in ImSucc^{tight}(s) \end{cases}$$

- $f^D(x) = f_t^D(x)$ *is called* delay-function *if*

$$f_j^D(x) := \begin{cases} \bigvee_{(T,j)\in A} \left(\bigwedge_{i\in T}(x_i \vee f_i^D(x))\right) & \text{for } j \notin ImSucc^{tight}(s) \\ \text{FALSE} & \text{for } j \in ImSucc^{tight}(s) \end{cases}$$

- $f^B(x) = x_t \wedge f_t^B(x)$ *is called* bulk-function *if*

$$f_j^B(x) := \begin{cases} \bigwedge_{(T,j)\in A} \left(\bigvee_{i\in T}(x_i \wedge f_i^B(x))\right) & \text{for } j \notin ImSucc^{tight}(s) \\ \text{TRUE} & \text{for } j \in ImSucc^{tight}(s) \end{cases}$$

Note that those functions are well defined. Each of them is defined on the hypergraph $H(N^{tight})$ of the tight sub-network of N. Network N^{tight} does not contain any cycles, therefore also its hypergraph is acyclic. Thus there cannot occur any cyclic dependencies of two functions f_i and f_j depending on i and j and their predecessors.

First, we want to have a look onto the properties of the defined functions. For the cut- and the delay-function f is true if f_t is true, whereas in the path and the bulk case, f is true if x_t is true and function f_t is true. This function f_j, depending on j works on the tails of the incoming arcs of j and can be called a predecessor condition. Looking at the predecessor condition of the immediate successors of s, there is again a difference between the cut- and delay-function on one side and the path- and bulk-function on the other side. For the cut- and delay-function, the predecessor condition of the immediate successors of s is not fulfilled, while in the path and bulk case it is.

Let us consider the cut-function in detail now. $f^C(x)$ is true if the predecessor condition of t is fulfilled. For a variable x_j, the predecessor condition f_j^C is true if for all incoming hyperedges (T, j) there exists one element $x_i \in T$ in the tail of the hyperedge such that either x_i is true or its predecessor condition is true. Let us compare this to the requirements of a cut-critical set to be able to shorten the makespan. The meaning should be interpreted in the way that, if a variable is true, then its processing time is decreased, and if its predecessor condition is true, then its start time is decreased. Both actions result in a *decrease* of its completion time. A variable that is false corresponds to a job, where the processing time is not changed. So obviously, the makespan is decreased, if the start time of t is decreased, which corresponds to f_t^C being true, and thus also the cut-function is true.

To be able to decrease the start time of a job, the completion time of all its predecessors has to be decreased. This is obtained by either shortening the processing time or decreasing the start time of one element of every incoming hyperedge. Remember that a hyperedge corresponds to either an edge, then the completion time of the tail of the edge has to be decreased by one of the possible actions. Or the hyperedge corresponds to an OR-node, in which case it suffices to complete one job in its predecessor set earlier. The predecessor condition for the immediate successors of s is not fulfilled, corresponding to the fact that the completion time of s cannot be decreased. It should be reasonable that an inclusion-minimal fulfilling assignment coincides with a cut-critical set.

We can follow the same arguments for true meaning an *increase* in completion time, to come up with a correspondence of an inclusion-minimal fulfilling truth assignment for the delay-function and a delay-critical set. We will give a formal proof of this later.

Now consider the path-function $f^P(x) = x_t \wedge f_t^P(x)$. The function f^P is true if x_t and its predecessor condition are true. Now the value true should be interpreted as *no* change in completion time, and thus no change in start and processing time. For a false variable, the processing time is decreased. For a variable x_j, the predecessor condition is fulfilled, if there exists one incoming hyperedge (T, j) such that for every element i in the tail of this hyperedge, x_i and its predecessor condition is fulfilled. For the immediate successors of s, the predecessor condition is always satisfied, as the completion time of s does not change. From this we conclude that an inclusion-minimal satisfying truth assignment for f^P has to contain x_t and for every variable it contains, it also has to fulfill its predecessor condition. If we recall Definition 2.3.14 of an OR-closed set, we see that the path-function exactly describes property ($\vee$). Thus we can conclude that an inclusion-minimal fulfilling assignment of the path-function corresponds to an OR-closed set, which itself is a path-critical set.

For the last case, the bulk-function, we can follow the argumentation of the path-function. This time we assume that a false variable has an increased processing time. The bulk-function describes property ($\wedge$) and we get that an inclusion-minimal fulfilling assignment of the bulk-function corresponds to an AND-closed set and thus also a bulk-critical set.

Let us make one more remark about those four functions, before we go into the details. Consider a slightly modified version of the cut- and the delay-function. If we fix $x_t = \text{FALSE}$, we can rewrite $f^C(x) = f_t^C(x) = x_t \vee f_t^C(x)$ and also $f^D(x) = f_t^D(x) = x_t \vee f_t^D(x)$ without changing the value of the function. Now

we apply negation to the predecessor conditions of the cut-function and get

$$\begin{aligned}\neg f_j^C(x) &= \begin{cases} \neg\left(\bigwedge_{(T,j)\in A}\left(\bigvee_{i\in T}(x_i \vee f_i^C(x))\right)\right) & \text{for } i \notin ImSucc^{tight}(s) \\ \neg\,\text{FALSE} & \text{for } i \in ImSucc^{tight}(s) \end{cases} \\ &= \begin{cases} \bigvee_{(T,j)\in A}\left(\bigwedge_{i\in T}(\neg\, x_i \wedge \neg f_i^C(x))\right) & \text{for } i \notin ImSucc^{tight}(s) \\ \text{TRUE} & \text{for } i \in ImSucc^{tight}(s) \end{cases} \\ &= f_j^P(\neg\, x)\end{aligned}$$

For the cut-function itself it follows that

$$\neg f^C(x) = \neg f_t^C(x) = \neg\, x_t \wedge \neg f_t^C(x) = f^P(\neg\, x)$$

In words this means that if an assignment does not fulfill the cut-function, then its negation fulfills the path-function. We can do the same calculation for the delay function and get

$$\neg f^D(x) = \neg\, x_t \wedge \neg f_t^D(x) = f^B(\neg\, x), \text{ with } \neg f_j^D(x) = f_j^B(\neg\, x).$$

So again, if an assignment does not fulfill f^D, then its complement is a satisfying truth assignment for f^B.

We now want to state formally what we have explained so far.

Lemma 2.4.6. *Let $H(N^{tight})$ be the hypergraph of the tight sub-network of a given AND/OR-network $(N = (V \cup W, E), p)$. Then the inclusion-minimal fulfilling assignments of the path-function [bulk-function] on $H(N^{tight})$ correspond one to one to path-critical [bulk-critical] sets of N.*

Proof. The path-function [bulk-function] exactly describes property ($\vee$) [($\wedge$)] of OR-closed [AND-closed] sets of the AND/OR-network N. Thus an inclusion-minimal satisfying assignment of the path-function [bulk-function] corresponds directly to an OR-closed [AND-closed] set. Theorem 2.3.17 [Theorem 2.3.18] then concludes the proof. □

In the cut- and delay-critical case, we do not have this structural description directly corresponding to the respective functions, so we have to do a little bit more to prove the following lemma.

Lemma 2.4.7. *Let $H(N^{tight})$ be the hypergraph of the tight sub-network of a given AND/OR-network $(N = (V \cup W, E), p)$. Then the inclusion-minimal fulfilling assignments of the cut-function [delay-function] on $H(N^{tight})$ correspond one to one to cut-critical [delay-critical] sets of N.*

Proof. To prove this lemma, we have to show that a critical set corresponds to a fulfilling assignment and that a fulfilling assignment has property (C) or (D) respectively. The inclusion-minimality then again follows by the standard argument from the inclusion-minimality of the sets themselves. First we consider the cut-critical case.

Let $U^C \subseteq V$ be a cut-critical set of (N, p) and suppose that it does not correspond to a fulfilling assignment for the cut-function f^C of N. From the calculations above we know that $\neg f^C(x) = f^P(\neg x)$ and thus the complement $V \setminus U^C$ of the cut-critical set contains a fulfilling assignment for the path-function. We have stated in Lemma 2.4.6 that a satisfying truth assignment for the path-function corresponds to a path-critical set. It follows that there exists a path-critical set in the complement $V \setminus U^C$ of the cut-critical set U^C, a contradiction to property (C) of U^C. Now consider a set U corresponding to an inclusion-minimal fulfilling assignment for the cut-function and suppose that it does not have property (C). Then, the complement of U, $V \setminus U$, has property (P) in (N, p). By Lemma 2.4.6 it follows that $V \setminus U$ is a satisfying assignment for the path-function. As $f^P(\neg x) = \neg f^C(x)$ we get a contradiction to U being a fulfilling assignment for f^C. Note that we strongly exploit the monotonicity of the functions in those arguments.

Following the same scheme, let $U^D \subseteq V$ be a delay-critical set of (N, p) and suppose that it does not correspond to a fulfilling assignment for the delay-function f^D of N. Then with $\neg f^D(x) = f^B(\neg x)$ we get that the complement $V \setminus U^D$ contains a fulfilling assignment for the bulk-function. By Lemma 2.4.6 it follows that $V \setminus U^D$ contains a bulk-critical set, in contradiction to property (D) of the delay-critical set U^D. For the other direction let U correspond to an inclusion-minimal satisfying truth assignment for the delay-function. For the sake of contradiction, suppose that it does not have property (D). Then the complement $V \setminus U$ has property (B) and thus contains a fulfilling assignment for the bulk-function. As $f^B(\neg x) = \neg f^D(x)$, this contradicts the fact that U is supposed to fulfill the delay-function, altogether concluding the proof. □

Let us now come to monotone Boolean functions in general. We want to have a look at inclusion-minimal satisfying truth assignments and the variables contained in them. By the monotonicity of monotone Boolean functions it follows that a variable x_j not contained in any inclusion-minimal fulfilling assignment is not important: any satisfying truth assignment for a function f containing x_j still fulfills f, if we remove x_j from the assignment. The value of an assignment does not depend on the value of x_j. It might be useful to be able to detect variables with this property, as they might help to simplify the function under consideration for example. We want to address this question now. Therefore we first define an unimportant variable.

Definition 2.4.8. *Let f be a monotone Boolean function on n variables $x = (x_1, \ldots, x_n)$. A variable x_j is called* redundant *if no inclusion-minimal satisfying truth assignment for f depends on the value of x_j.*

For the main theorem we assume that a monotone Boolean function is given as a function over the basic operators $\vee$ and $\wedge$. This is not a restriction as it is well known that the functions computable using just the operators $\vee$ and $\wedge$, and parentheses of course, are exactly the monotone Boolean functions. For any further details we again refer to Wegener (1987), for example.

Theorem 2.4.9. *Given a monotone Boolean function f over the operators $\vee$ and $\wedge$ on n variables $x = (x_1, \ldots, x_n)$ and a specific variable x_j, then it is NP-complete to decide whether x_j is not redundant.*

Proof. To prove this theorem we reduce the question "j cut-critical?" to the question "x_j contained in an inclusion-minimal fulfilling assignment?". From the construction used in the proof of Theorem 2.4.3 to establish the NP-hardness of checking whether a job is cut-critical, we know that this problem is already hard for AND/OR-networks of a very restricted structure. Let an AND/OR-network $(N = (V \cup W, E), p)$ as in the proof of Theorem 2.4.3 be given. We know that it is NP-complete to decide whether job $j \in V$ is cut-critical. By construction, the whole network is tight. We set up the cut-function as in Definition 2.4.5. This can be done in polynomial time due to the special structure of the given AND/OR-network N. By Lemma 2.4.7 we know that an inclusion-minimal satisfying assignment for the cut-function corresponds to a cut-critical set and vice versa. Therefore, j is contained in a cut-critical set (and thus is cut-critical) if and only if x_j is not redundant. As it is NP-hard to decide whether j is cut-critical by Theorem 2.4.3, this shows that it is NP-hard to decide if x_j is not redundant. On the other hand, an inclusion-minimal satisfying truth assignment containing variable x_j is an easily checkable certificate for x_j to be not redundant. We conclude that it is NP-complete to decide for a specific variable x_j, whether x_j is not redundant. □

To the authors' knowledge, the question about redundant variables in monotone Boolean functions has not been answered before. Thus the presented complexity result gives a new insight into the hardness of monotone Boolean functions. Many publications can be found on minimum, in contrast to inclusion-minimal satisfying truth assignments, or minimum unsatisfied clause problems on monotone as well as general Boolean functions. It is known that finding an inclusion-minimal fulfilling assignment is easy for monotone Boolean functions, in correspondence to our result of Theorem 2.4.1 about critical sets. Finding a minimum fulfilling assignment, in contrast is NP-hard for monotone functions,

see Khanna, Sudan, and Trevisan (1997) for example. Khanna et al. consider constraint satisfaction problems and classify them by certain properties. They prove for both problems, the minimum satisfying truth assignment problem and the minimum unsatisfied clause problem, that the hardness of the problem is completely determined by the properties the formula has.

2.5 Criticality in Specially Structured AND/OR-Networks

After this little excursion into logic, we want to concentrate on AND/OR-networks again. We have seen that there exist four interpretations of criticality in AND/OR-networks, where in each case, two of them form a pair of blocking clutters and by this are equivalent. We have also seen in the two little example networks presented in Figure 2.2 that the two pairs are not equivalent in general. It is natural to ask, whether there are cases, in which the four notions fall together. Looking at the examples in Figure 2.2 again, it seems as if edge $(1, 4)$ plays a special role compared to other edges. Is it possible to classify the structure of an AND/OR-network in which the definitions are equivalent? We are able to partially answer this question. First we have to define two structures on AND/OR-networks, which are well known in standard graphs. An AND/OR-network $N = (V \cup W, E)$ is an ***in-tree*** , if every vertex $v \in V \cup W$ has exactly one direct successor. Remember that we have assumed this for the OR-nodes in the beginning. For an in-tree, we also impose this restriction onto the AND-nodes. The second structure probably is better known from order theory. A graph $G = (V, E)$ with $V = \{u, v, x, y\}$ and $E = \{(u, v), (x, y), (u, y)\}$ is called ***N***. Note that edge (x, v) is not allowed to be present. We extend this definition to AND/OR-networks by allowing the vertex set to contain AND- and OR-nodes. Thus, an AND/OR-network $N' = (V' \cup W', E')$ is called an N, if $V' \cup W' = \{u, v, x, y\}$ and $E' = \{(u, v), (x, y), (u, y)\}$. An AND/OR-network $N = (V \cup W, E)$ is said to be ***N-free*** if no four element vertex set $U = \{u, v, x, y\} \subseteq V \cup W$ induces an N. For an example look at the two networks in Figure 2.2. In N^a, the nodes 1, 2, 3, and w form an N, while in N^b the nodes 1, 2, 3, and 4 form an N.

Theorem 2.5.1. *Given an AND/OR-network $(N = (V \cup W, E), p)$ with tight sub-network N^{tight}. Let $T = (V^T \cup W^T, E^T)$ be the sub-network of N^{tight} containing only the tight s-t-paths. If $T \setminus \{s\}$ is an in-tree, then every job j of V^T is cut-, path-, delay-, and bulk-critical.*

Of course this lemma implies that if T without s is an in-tree, then the four definitions of a critical job are equivalent.

Proof. Let $T = (V^T \cup W^T, E^T)$ without s be an in-tree. Note that it suffices to consider the sub-network consisting of the tight s-t-paths. For any vertex v holds

that there is a tight path from s to v by the definition of the earliest start schedule. But if a vertex v is not in $V^T \cup W^T$, then there is no tight path from v to t as otherwise v would be on a tight s-t-path. Thus for every v-t-path, there exists at least one edge that is not tight. Consequently, if v is a job, neither prolonging nor shortening the processing time of v by a small $\varepsilon < slack(N, p)$ can influence the makespan. That means that v cannot be cut- or delay-critical and by Corollary 2.3.4 and Corollary 2.3.6 it follows that v cannot be path- or bulk-critical either.

We will prove that a given job $j \in V^T$ is cut-critical by constructing a cut-critical set U^C containing j and equivalently prove that j is delay-critical by constructing a delay-critical set U^D containing j. From this and the Corollaries 2.3.4 and 2.3.6 it then follows that every job is critical in every of the four senses.

First we concentrate on the cut-critical case. We use the hypergraph notation of T. So let $H(T) = (V^T, A)$ be the hypergraph of T. We consider the unique path P_{jt} from j to t in $H(T)$, $P_{jt} = (j = v_0, a_1, v_1, \dots a_k, v_k = t)$. We construct a cut-critical set U^C along the path P_{jt} starting with $U^C = \{j\}$. We consider each hyperedge a_i on the path, $i = 1, \dots, k$, starting from $i = 1$. Let $a_i = (T_i, v_i)$, then for every hyperedge $(T_a, v_i) \neq a_i$, going into v_i, we select one element $x_a \in T_a$ and include x_a into U^C. Note that there does not exists an $x_a \in T_a$ such that $x_a \in T_b$ for some other hyperedge b, due to the in-tree structure of T. We claim that for every v_i, $i = 1, \dots, k$, U^C corresponds to a fulfilling assignment for the predecessor condition $f^C_{v_i}$ of the cut-function. This yields that U^C is a truth assignment for the cut-function f^C itself. Recall that for some v_i, $f^C_{v_i}$ is true if for every incoming hyperedge, for one of the elements x in the tail of the hyperedge either x or its predecessor condition is true. Look at v_1 first. For the incoming hyperedge a_1 on P_{jt}, we have that j is in U^C and thereby fulfills the predecessor condition for a_1. For every other incoming hyperedge $a = (T_a, v_1)$, we have included exactly one element x from its tail T_a in U^C. Thus $f^C_{v_1}$ is fulfilled. By induction, for every later node v_i on P_{jt}, we have the following. The incoming hyperedge $a_i = (T_i, v_i)$ which is on the path is satisfied by the predecessor condition of the node in its tail which is on the path, $v_{i-1} \in T_i \cap P_{jt}$. For every other hyperedge $a = (T_a, v_i)$ we have included exactly one element of its tail T_a into U^C. Thus, for every i, $i = 1, \dots, k$, the predecessor condition of v_i is fulfilled proving that the set U^C provides a truth assignment for f^C. It is left to show that U^C is inclusion-minimal. Suppose this is not the case, then there exists some $u \in U^C$ such that $U^C \setminus \{u\}$ also provides a fulfilling assignment for f^C. By construction, this u has to be in some tail of a hyperedge going into one of the v_i on the path P_{jt}. Consider a $u \in U^C$ that is in the tail T_a of a hyperedge $a = (T_a, v_\ell)$ going into v_ℓ with the highest index ℓ. As $H(T)$ is an in-tree $T_a \cap T_b = \emptyset$ for any two hyperedges a and b. For every job on P_{jt} we have chosen exactly one element of the tail of every incoming hyperedge not on P_{jt} to be in U^C. Thus $U^C \setminus \{u\}$ does not contain any of the

elements of T_a for hyperedge a going into v_ℓ. To fulfill $f^C_{v_\ell}$, then the predecessor condition for some element in T_a has to be fulfilled. But this is not possible, as no job $v \in U^C$, which is true, and no $v_i \in P_{jt}$, for which the predecessor function is fulfilled, is in more than one tail due to the in-tree structure of $H(T)$. This yields that the predecessor function of v_ℓ cannot be fulfilled if we remove u from U^C. For every job v_i on the path, the incoming hyperedge a_i which is also on the path is only fulfilled by either v_0, if $i = 1$ or by the predecessor function of its direct predecessor on the path, v_{i-1}. Thus, if the predecessor function for an element on the path is not fulfilled, this carries through to the end t of the path. This proves that every $j \in V^T$ is cut-critical.

Let us consider the delay-critical case now. We do a similar construction to get a delay-critical set U^D containing j. We again use the hypergraph notation of T and consider the unique path $P_{jt} = (j = v_0, a_1, v_1, \dots a_k, v_k = t)$ from j to t. At the beginning, let U^D only contain j, $U^D = \{j\}$. We walk up the path P_{jt} and consider each hyperedge a_i on the path, $i = 1, \dots, k$, starting with $i = 1$. Let $a_i = (T_i, v_i)$ be the hyperedge on P_{jt} under consideration. We include every node in $T_i \setminus \{v_{i-1}\}$ into U^D. Note that this set is empty if $a_i \in A^E$, that is if a_i corresponds to an edge. We show that we have constructed a set U^D providing a fulfilling assignment for the delay-function f^D. We claim that the predecessor function $f^D_{v_i}$ is fulfilled for every v_i on the path, thus fulfilling the delay-function. For every v_i on the path, there exists a hyperedge, namely a_i on P_{jt}, such that for every element x in the tail of a_i, either x is in U^D, or its predecessor function f^D_x is fulfilled. This is true by construction, as for every element x in the tail of a hyperedge on the path, either x is on the path but not equal to j, then its predecessor function is true, or it is not on the path or equal to j, then it is included in U^D. The inclusion-minimality of set U^D again follows from the in-tree structure of $H(T)$ and the construction. The in-tree structure of $H(T)$ assures that $T_a \cap T_b = \emptyset$ for the tails of any two hyperedges a and b. By construction, the predecessor function of every v_i on P_{jt} is exactly fulfilled by the incoming hyperedge a_i which is on P_{jt}. This yields that U^D is also inclusion-minimal and concludes the proof. □

An in-tree is a very restricted structure, as can be seen in the fact that not only the four definitions of criticality are equivalent, but every vertex on a tight s-t-path is critical, in correspondence to the standard network case. In addition, it is easy to decide whether a given job is critical or not.

Corollary 2.5.2. *Given an AND/OR-network $(N = (V \cup W, E), p)$, let $T = (V^T \cup W^T, E^T)$ be the sub-network of tight s-t-paths. If T without s is an in-tree, we can decide in time polynomial in $size(N)$ whether a job $j \in V$ is cut-, path-, delay-, and bulk-critical.*

As a last remark on the tree-structure we want to mention that it is obviously easy to check whether a given AND/OR-network N has the in-tree property. We simply have to calculate the earliest start schedule and remove every non-tight edge and vertex. We are left with an acyclic directed graph, where we can identify the set of all nodes that are on tight s-t-paths by a simple backward breadth-first search starting from t. During the search we can also check for the tree property, that is, whether every vertex has just one direct successor.

In the literature a lot of work about partially ordered sets and special classes of partially ordered sets can be found. We want to restrict to two references, both applying results and methods of order theory to combinatorial optimisation problems. In Möhring (1989) the focus is laid on special classes of ordered sets that admit efficient algorithms for otherwise intractable problems. The computational tractability of those classes in most cases emanates from strong structural properties not shared by general partial orders. A tree is such a restricted structure yielding a substantial simplification of the complexity of critical jobs as we have seen. An order-theoretic approach to scheduling can be found in Möhring and Radermacher (1989). In this paper, order-theoretic methods and results are successfully applied to scheduling problems over partially ordered sets. Again special structures of the partial orders are strongly exploited for the design of efficient algorithms.

We now want to look at a weaker structural property on AND/OR-networks, where we can still get a stronger relation between critical jobs than in the general case. Unfortunately the relation is weaker than in the in-tree case. A crucial property of the in-tree needed for the inclusion-minimality of the sets constructed was that $ImPred^{tight}(u) \cap ImPred^{tight}(v) = \emptyset$ for any two vertices $u, v \in V^T$. Thus any job chosen in the predecessor set of one job, cannot fulfill any predecessor condition for any other job. Something similar holds for an N-free network.

For this we need the following property of N-free partial orders which can be found in Möhring (1989), for example, as well.

Theorem 2.5.3. *Let $P = (V, A)$ be a partial order in transitively reduced presentation, then the following properties of P are equivalent.*

1. *P is N-free.*
2. *For any two $u, v \in V$ either $ImPred(v) = ImPred(u)$ or $ImPred(v) \cap ImPred(u) = \emptyset$.*
3. *For any two $u, v \in V$ either $ImSucc(v) = ImSucc(u)$ or $ImSucc(v) \cap ImSucc(u) = \emptyset$.*

In N-free partial orders, the predecessor sets of two nodes are not necessarily disjoint, like for a tree, but they are either disjoint or they are the same. We use this property to prove the next theorem.

Theorem 2.5.4. *Given an AND/OR-network $N = (V \cup W, E)$ with processing time vector p, let N^{tight} be the corresponding tight sub-network. If N^{tight} is N-free, then, if a job $j \in V$ is path-critical then j is also bulk-critical, but not conversely.*

Proof. Let $N^{tight} = (V^{tight} \cup W^{tight}, E^{tight})$ be N-free and suppose there exists some job $j \in V$ such that j is path-critical but not bulk-critical. Among all jobs in V with this property, consider j with maximal start time S_j. As j is path-critical, there exists some path-critical set U^P containing j. The set U^P corresponds to an inclusion-minimal truth assignment for the path-function f^P. Thus, there exists a job $k \in U^P$ and an incoming hyperedge $a = (T_a, k)$ such that $j \in T_a$ and the predecessor condition of k, f_k^P, is fulfilled by the hyperedge a. Such a job and hyperedge has to exist as otherwise, j would not be necessary for the inclusion-minimal truth assignment corresponding to U^P. We know that k is path-critical and bulk-critical by the maximal choice of j. Thus there exists a bulk-critical set U^B containing k. The set U^B corresponds to an inclusion-minimal satisfying assignment for the bulk-function f^B, and therefore, the predecessor condition f_k^B of k has to be fulfilled. This means that for every incoming hyperedge (T, k), there exists one $i \in T$, such that i and its predecessor condition is true. This has to hold for $a = (T_a, k)$ as well. We can conclude that a has to correspond to an OR-node w_a, as otherwise j would be in U^B automatically. To fulfill f_k^B, there has to exist some $i \in T_a \setminus \{j\}$ and $i \in U^B$. We have to go back to the standard AND/OR-network notation now. Consider the set of direct successors of i that are in the closure of U^B, $Y := ImSucc^{tight}(i) \cap \overline{U^B}$. Then $w_a \in Y$ and by Theorem 2.5.3, $j \in ImPred^{tight}(u)$ for all $u \in Y$. We get that $Y \subseteq W$, as an AND-node in $ImSucc^{tight}(j) \cap \overline{U^B}$ would again force j to be in U^B. Define X to be the set of all tight predecessors of j, including j itself, by definition. Consider the set $U := (U^B \setminus \{i\}) \cup X$. As U^B is an inclusion-minimal truth assignment for the bulk-function, we know that there exist a hyperedge (T, u) with $i \in T$ and $u \in U^B$, such that the predecessor condition of u is not fulfilled by $U^B \setminus \{i\}$. W. l. o. g., we assume that $a = (T_a, k)$ has this property. In set U, the predecessor condition of k can be fulfilled by j and some vertices in X, to fulfill the predecessor condition of j. Note that for every hyperedge $b = (T_b, u_b)$ with $i \in T_b$ and $u_b \in U^B$, we have already seen that b has to correspond to some OR-node w_b. From the N-freeness it directly follows that $j \in T_b$ and thus X, including j, can fulfill the predecessor condition for every u_b, for which it is not fulfilled by $U^B \setminus \{i\}$. Bringing these arguments together we conclude that there exists a bulk-critical set $U^* \subseteq U$. If we can show that $j \in U^*$, we are finished.

Claim: Let $U^* \subseteq U$ be some bulk-critical set, then $j \in U^*$. By construction it is clear that if $k \in U^*$, then also $j \in U^*$. So suppose that $k \notin U^*$. Let Z be the set of tight successors of k. Consider ℓ with maximal start time S_ℓ such

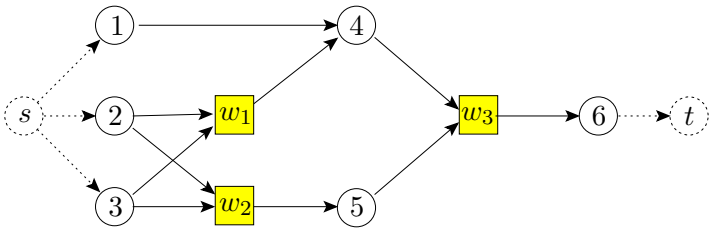


Figure 2.8: AND/OR-network N with $p = (1, \ldots, 1)$. Job 1 is bulk-critical but not path-critical.

that $\ell \in Z \cap U^B$ but $\ell \notin U^*$, $\ell = k$ possible. By the inclusion-minimality of U^B with respect to the property of fulfilling the path-function, there exist a hyperedge $c = (T_c, v)$ with $\ell \in T_c$, $v \in U^B$ and $v \in U^*$, such that the predecessor condition for v does not get fulfilled by $U^B \cap U^*$. As U^* itself corresponds to a truth assignment of the bulk-function, there exists an $x \in (U^* \setminus U^B) = (U^* \cap X)$ fulfilling the predecessor condition for v. This means that there exists either a tight edge (x, v) or two tight edges (x, w) and (w, v), where w is an OR-node having processing time zero. In any case, we get that $S_v = S_x + p_x$, where x is a predecessor of j, k a tight successor of j and v a tight successor of k, but not k itself. This can only be possible if $p_k = 0$, a contradiction to our assumption of strictly positive processing times for the jobs. Thus $k \in U^*$, implying that $j \in U^*$.

To conclude the proof, we give a counterexample for the reverse direction in Figure 2.8. In the presented AND/OR-network with unit processing times, there is only one path-critical set, $\{2, 3, 4, 5, 6\}$. On the other hand, set $\{1, 2, 4, 6\}$, containing vertex 1, is bulk-critical. The network is clearly N-free, but job 1 is bulk-critical and not path-critical. □

The next corollary follows directly from the above Theorem 2.5.4 together with two Corollaries 2.3.4 and 2.3.6 about the equivalence of cut- and path-critical jobs and delay- and bulk-critical jobs.

Corollary 2.5.5. *Let (N, p) be an AND/OR-network with tight sub-network N^{tight}. If N^{tight} is N-free, then every cut-critical job $j \in V$ is also delay-critical, but not the other way around.*

The AND/OR-network in Figure 2.8 is N-free as we have designed it, yet there is a certain asymmetry in the network. If we remove the three OR-nodes and therefore include edges from their direct predecessor to their direct successor, we get two N, induced by $\{1, 4, 2, 5\}$ and $\{1, 4, 3, 5\}$. Although we can only give a negative result, we want to study a structure of an AND/OR-network inspired by these considerations. We extend the notion of N-freeness to hypergraphs in a straightforward matter.

Definition 2.5.6. *A hypergraph $H = (V, A)$ is called* N-free*, if and only if, for any two nodes $u, v \in V$ either*

$$\left(\bigcup_{(X,u) \in A} X = \bigcup_{(Y,v) \in A} Y \right) \text{ or } \left(\bigcup_{(X,u) \in A} X \cap \bigcup_{(Y,v) \in A} Y = \emptyset \right)$$

A hypergraph is N-free, if for any two nodes, the set of direct predecessors is either the same or it is disjoint. This definition corresponds to the property of standard N-free graphs. It is easy to prove that the following holds.

Lemma 2.5.7. *Given some vertex $v \in V$ of a hypergraph $H = (V, A)$, let $HypSucc(v) := \{u \in V \mid (T, u) \in A,\ v \in T\}$ denote the set of direct successors of v in the hypergraph. Then, hypergraph H is N-free, if and only if for any two nodes $u, v \in V$ it holds that either*

$$\Big(HypSucc(u) = HypSucc(v) \Big) \text{ or } \Big(HypSucc(u) \cap HypSucc(v) = \emptyset \Big)$$

Proof. We have to prove that the characterisation of N-free in the definition implies the characterisation of the lemma and vice versa. So for the sake of contradiction suppose that there exist $u, v \in V$ such that

$$\Big(HypSucc(u) \cap HypSucc(v) \neq \emptyset \Big) \text{ and } \Big(HypSucc(u) \neq HypSucc(v) \Big).$$

Without loss of generality, assume $HypSucc(u) \setminus HypSucc(v) \neq \emptyset$. Then there exists $x, y \in V$ such that

$$\begin{aligned}
&\Big(x \in HypSucc(u) \setminus HypSucc(v) \Big) \text{ and } \Big(y \in HypSucc(u) \cap HypSucc(v) \Big) \\
\Rightarrow &\left(\Big(u \in \bigcup_{(X,x) \in A} X \Big) \text{ but } \Big(v \notin \bigcup_{(X,x) \in A} X \Big) \right) \text{ and} \\
&\left(\Big(u \in \bigcup_{(Y,y) \in A} Y \Big) \text{ and } \Big(v \in \bigcup_{(Y,y) \in A} Y \Big) \right) \\
\Rightarrow &\left(\bigcup_{(X,x) \in A} X \neq \bigcup_{(Y,y) \in A} Y \right) \text{ and } \left(\bigcup_{(X,x) \in A} X \cap \bigcup_{(Y,y) \in A} Y \neq \emptyset \right)
\end{aligned}$$

This contradicts the definition of N-free. The reverse implication follows by exactly the same arguments and is omitted. □

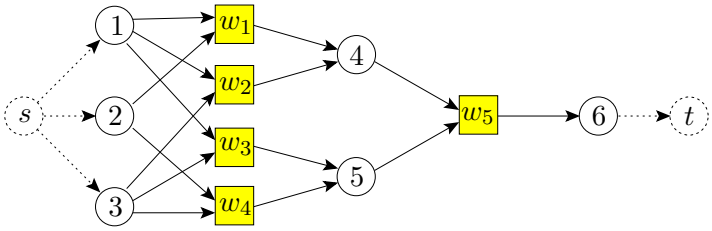


Figure 2.9: AND/OR-network N with $p = (1, \ldots, 1)$. Job 2 is bulk-critical but not path-critical.

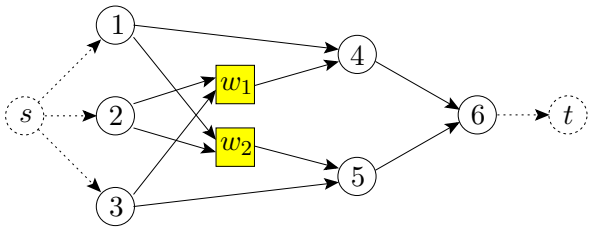


Figure 2.10: AND/OR-network N with $p = (1, \ldots, 1)$. Job 2 is path-critical but not bulk-critical.

Is seems to be likely that the different critical notions are equivalent if the hypergraph of the tight sub-graph of an AND/OR-network is N-free. Unfortunately this is not the case. As a matter of fact, we do not even have one implication, as in the other case, where N itself is N-free.

Proposition 2.5.8. *For an AND/OR-network $(N = (V \cup W, E), p)$, let $H(N^{tight})$ be the hypergraph of the tight sub-network of N. Even if $H(N^{tight})$ is N-free, for some vertex $j \in V$, j path-critical does not imply that j is bulk-critical and j bulk-critical does not imply that j is path-critical.*

Proof. The proof is done by giving appropriate counter-examples, of course. Look at the network in Figure 2.9 with processing time vector $p = (1, \ldots, 1)$. The only path-critical set in the network is $U^P = \{1, 3, 4, 5, 6\}$. As U^P does not contain job 2, it is not path-critical. In contrast to this, for example the set $\{2, 3, 4, 6\}$ is bulk-critical, proving that job 2 is bulk-critical.

Now consider the AND/OR-network in Figure 2.10 again with unit processing times on the jobs. In this network, the set $U^B = \{1, 3, 4, 5, 6\}$ is the only bulk-critical set. Job 2 is not contained in U^B and thus not bulk-critical. On the other hand, $2 \in \{2, 3, 4, 6\}$, which is a path-critical set. This proves that job 2 is itself path-critical. □

In summary we can say that the four critical notions are not equivalent in general. But even in already structurally quite special AND/OR-networks they yield

different sets of critical jobs. We were able to classify AND/OR-networks where for a job the property of being cut- and path-critical implies the property of being delay- and bulk-critical but not conversely. In addition we identified a property of AND/OR-networks implying that all the four definitions are equivalent. Although this property implies a very restricted structure, it has the great advantage that it can be checked easily and if it is present, then the set of critical jobs can be computed efficiently as well.

2.6 A Quantifying Analysis of Critical Sets

In the last section of this chapter we want to make some sort of sensitivity analysis in AND/OR-networks. Therefore we want to quantify the amount of change in the makespan subject to the change of processing times. We prove that the cut- and delay-critical sets have a straight impact on the makespan. Any change of processing time of a critical set by some small amount implies a change of the makespan by exactly the same amount. This will be shown in the first subsection. In the subsection thereafter we will shortly consider the time-cost trade-off problem. In this problem, processing times are assumed to be variable and to imply certain costs. For every job, a range of possible processing times is given together with a cost function on the processing time. The cost function is assumed to be non-increasing, corresponding to the intuition that it should be more expensive, if a job is completed more quickly.

2.6.1 Changing the Makespan

In this section we want to make a quantitative analysis of critical sets. In particular, we want to determine the amount of change in the makespan depending on the amount of change of the processing times of critical jobs. As there is no change in the makespan for path- and bulk-critical sets, we concentrate on cut- and delay-critical sets in the following.

Theorem 2.6.1. *For a given AND/OR-network $(N = (V \cup W, E), p)$, let U^C be a cut-critical and U^D a delay-critical set. Then, there exists a $\delta > 0$ such that $C_{\max}(S^{[U^C-\varepsilon]}) = C_{\max}(S) - \varepsilon$ and $C_{\max}(S^{[U^D+\varepsilon]}) = C_{\max}(S) + \varepsilon$ for every $0 < \varepsilon \leq \delta$.*

Proof. Let U^C be a cut-critical and U^D a delay-critical set for (N, p). According to Lemma 2.3.10 there exists a $\delta > 0$ such that $S_j^{[U^C-\varepsilon]} = S_j$ for every $j \in U^C$

and $S_j^{[U^D+\varepsilon]} = S_j$ for every $j \in U^D$, for all $0 < \varepsilon \leq \delta$. We take

$$0 < \delta < \frac{slack(N,p)}{|V|}$$

such that it fulfills Lemma 2.3.10 and keep this δ fixed in the following.

First we will pay attention to the cut-critical case. By contradiction, suppose that $C_{\max}(S^{[U^C-\varepsilon]}) \neq C_{\max}(S) - \varepsilon$ for some $0 < \varepsilon \leq \delta$. Then, there exists a node $v \in V \cup W$ with $C_v^{[U^C-\varepsilon]} \notin \{C_v, C_v - \varepsilon\}$. We choose such a node v with minimal completion time C_v. If v is not unique, we can take any of the choices if they are incomparable. If they are comparable, we prefer tight predecessors. Lemma 2.3.10 yields that $v \notin U^C$ and therefore $p_v^{[U^C-\varepsilon]} = p_v$. By the minimal choice of v it follows that $C_u^{[U^C-\varepsilon]} \in \{C_u, C_u - \varepsilon\}$ for every u with $C_u < C_v$. Since $S^{[U^C-\varepsilon]}$ is an earliest start schedule, there exists a tight direct predecessor u of v with $C_u^{[U^C-\varepsilon]} = S_v^{[U^C-\varepsilon]}$. By the choice of $\varepsilon \leq \delta < slack(N,p)/|V|$ and Lemma 2.2.7, $E^{tight}(S^{[U^C-\varepsilon]}) \subseteq E^{tight}(S)$ and thus u is also a tight direct predecessor of v with respect to schedule S. This yields a contradiction to our assumption, as due to the minimal choice of v it holds that $C_u^{[U^C-\varepsilon]} \in \{C_u, C_u - \varepsilon\}$ which implies that $C_v^{[U^C-\varepsilon]} \in \{C_v, C_v - \varepsilon\}$. This concludes the proof for cut-critical sets.

Let us switch to the delay-critical case now. By contradiction, suppose that $C_{\max}(S^{[U^D+\varepsilon]}) \neq C_{\max}(S) + \varepsilon$ for some $0 < \varepsilon \leq \delta$. Then, there exists a node $v \in V \cup W$ with $C_v^{[U^D+\varepsilon]} \notin \{C_v, C_v + \varepsilon\}$ and we again choose such a node v with minimal completion time C_v. In the case of equality, if the choices are comparable we prefer tight predecessors otherwise we can choose any of them. Lemma 2.3.10 yields $v \notin U^D$ and thus $p_v^{[U^D+\varepsilon]} = p_v$. By the minimal choice of v it holds that $C_u^{[U^D+\varepsilon]} \in \{C_u, C_u + \varepsilon\}$ for every u with $C_u < C_v$. Since $S^{[U^D+\varepsilon]}$ is an earliest start schedule, there exists a tight direct predecessor u of v with $C_u^{[U^D+\varepsilon]} = S_v^{[U^D+\varepsilon]}$. Moreover, from the choice of ε and Lemma 2.2.7 it follows that $E^{tight}(S^{[U^D+\varepsilon]}) \subseteq E^{tight}(S)$ and thus u is also a tight direct predecessor of v with respect to schedule S. By the minimal choice of v it holds that $C_u^{[U^D+\varepsilon]} \in \{C_u, C_u + \varepsilon\}$ which implies that also $C_v^{[U^D+\varepsilon]} \in \{C_v, C_v + \varepsilon\}$, a contradiction to our assumption. This concludes the proof for the delay-critical sets U^D and together with the first part completes the proof of the lemma. □

We have seen that the delay- and cut-critical sets have an unbiased impact on the makespan. Therefore they provide a useful tool for the project planner. A delay-critical set is critical in the sense that some small delay of the jobs in the set will directly delay the project completion time. It might be advisable for a planner to take care about the processing times of the jobs in a delay-critical

set. On the other hand, a cut-critical set offers the possibility of decreasing the project completion time efficiently, in the sense that a cut-critical sets guarantees a decrease of the makespan, if the processing times of its jobs, and only these, are decreased. The next section will present some more work in this area.

2.6.2 The Linear Time-Cost Trade-off Problem

We conclude this section by a short introduction into the *linear time-cost trade-off problem* on AND/OR-networks. As stated in Section 2.3.3, due to the big Max-Flow-Min-Cut gap in AND/OR-networks it is likely that the linear time-cost trade-off problem in AND/OR-networks is a hard problem. Indeed essentially the same reduction as for the Max-Flow-Min-Cut gap is used to establish the hardness of the linear time-cost trade-off problem. Most of the following results are taken from Stork (2001). In the linear time-cost trade-off problem the jobs are assumed to have variable processing times and involve costs, depending on the processing times. Intuitively this models the situation that it is more expensive to process a job quickly than it is to do it slowly. Formally, the dependence of the costs on the processing times of a job $j \in V$ is described by a non-increasing, non-negative, affine linear *cost function* $\kappa_j : [p_j^{min}, p_j^{max}] \to \mathbb{R}_{\geq}$. The value $p_j^{min} > 0$ expresses the shortest possible processing time of job j, whereas $p_j^{max} \geq p_j^{min}$ is the longest possible processing time. With $\mathbb{R}_{\geq}$ we denote the set of non-negative real numbers. The cost to run a job with processing time p_j is given by $\kappa_j(p_j)$. The problem is to find for all deadlines $T \geq 0$ a processing time vector p with $C_{\max}(p) \leq T$ that minimizes the total cost $\kappa(p) = \sum_{j \in V} \kappa_j(p_j)$. Note that the makespan depends on the processing time vector and we denote it by $C_{\max}(p)$. The solution to this problem consist of the so-called *time-cost trade-off curve* $B(T) := \min\{\kappa(p) \mid p^{min} \leq p \leq p^{max}, C_{\max}(p) \leq T\}$. Fulkerson (1961) and Kelley (1961) have independently discovered that $B(T)$ is monotone decreasing, piecewise linear, convex, and continuous if standard precedence constraints are imposed among the jobs. In addition, it can be constructed by a series of minimum cut computations. In the case of scheduling with AND/OR precedence constraints the scenery changes completely. We want to discuss the problems that arise on a little example.

Example 2.6.2. *Let $N = (V \cup W, E)$ consist of six jobs with three waiting conditions $(\{1,2\},4)$, $(\{1,3\},5)$, and $(\{2,3\},6)$. The network is displayed in Figure 2.3. The jobs have the following processing times and cost functions:*

job j	1	2	3	4	5	6
p_j^{min}	1	1	2	1	1	1
p_j^{max}	10	4	3	1	1	1

$$\kappa_1(p_1) = -1/5\, p_1 + 2$$
$$\kappa_2(p_2) = -p_2 + 4$$
$$\kappa_3(p_3) = -p_3 + 3$$

The jobs 4, 5, *and* 6 *have constant processing times and do not cause any costs.*

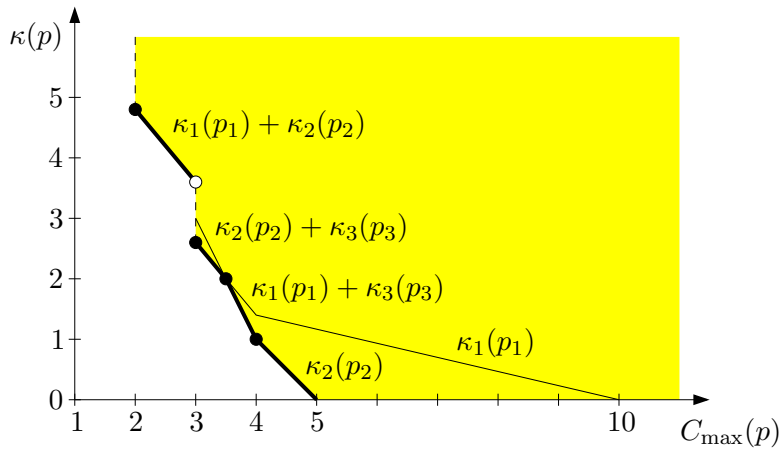


Figure 2.11: The non-continuous, non-convex time-cost trade-off curve for the problem presented in Example 2.6.2. The bold line marks the time-cost trade-off curve, the shaded area represents the set of feasible solutions.

The time-cost trade-off curve for the problem in the example is presented in Figure 2.11. The shaded area marks the feasible completion times of the project, bounded from below by the time-cost trade-off curve, which is drawn by a thick line.

We want to discuss the time-cost trade-off curve from right to left. We will concentrate on the jobs 1, 2, and 3, as the other jobs have constant unit processing times and cannot be decreased. To make the notation as short as possible, we will therefore write p as the triple (p_1, p_2, p_3) with the knowledge that the other processing times are fixed. First, $C_{\max}(p) = 5$ and all jobs take as long as they are allowed to, that is $p = (10, 4, 3)$. This processing time vector involves as few costs as possible, zero in our case. There is one cut-critical job, 2, which forms a cut-critical set on its own. Just like in the case of standard precedence constraints, we can decrease the processing time of job 2 from $p_2 = 4$ down to $p_2 = 3$. Now $p = (10, 3, 3)$ and $C_{\max}(p) = 4$ with total costs of $\kappa(p) = \kappa_2(p_2) = 1$. At this point, job 3 becomes cut-critical together with 2 in one cut-critical set $\{2, 3\}$. We again can decrease the processing times of the two jobs in the cut-critical set linearly involving total costs of $\kappa(p) = \kappa_2(p_2) + \kappa_3(p_3)$. Set $\{2, 3\}$ stays cut-critical until we have decreased the two jobs down to the minimal processing time of job 3, $p_2 = p_3 = p_3^{min} = 2$. This processing time vector, $p = (10, 2, 2)$, gives $C_{\max}(p) = 3$ with total cost $\kappa(p) = 3$. Although set $\{2, 3\}$ is cut-critical, this is not optimal, as we can see in the picture. Alternatively, we can reduce the processing time of job 1 down to $p_1 = 3$ at costs of $\kappa(3, 4, 3) = 7/5 > 1$. Note

that at the point where $p_1 = p_2 = 4$, each of the two jobs, 1 and 2, forms a cut-critical set on its own. When $p_1 = 3$, job 1 becomes cut-critical only together with job 3 and we have to decrease the processing times of both jobs together. We again can do that until p_1 and p_3 are equal to the minimal processing time of job 3. The processing time vector $p = (2, 4, 2)$ involves total cost of $\kappa(p) = 13/5 < 3$. Thus the optimal strategy of decreasing job processing times changes in the interval between $C_{\max} = 4$ and $C_{\max} = 3$. In fact, when $C_{\max} = 7/2$, we have that $\kappa(10, 5/2, 5/2) = \kappa(5/2, 4, 5/2) = 2$. At this point, the former optimal strategy of decreasing the processing times of the jobs in the critical set $\{2, 3\}$ is no longer optimal and we let job 2 become long again and shorten job 1 instead. Although this is done linearly from this point on, first job 1 has to be shortened by a large amount from $p_1 = 10$ down to $p_1 = 5/2$. The remarkable fact about this is that set $\{2, 3\}$ is cut-critical when we reach this point from the right but is no longer optimal. This means that the time-cost trade-off curve is continuous and actually concave in this area around $C_{\max} = 7/2$, but there does not exist a continuous change in the processing time vector realising the time-cost trade-off curve. In $C_{\max} = 3$, again a discontinuity appears, this time in both, the time-cost trade-off curve and the changes of the processing time vector. The processing time of job 3, which is critical together with job 1 cannot be shortened any more. Nevertheless, we can shorten job 2 again to $p_2 = 2$ and let job 3 be long again. Processing time vector $p = (2, 2, 3)$ involves costs of $18/5 > 13/5$, which is not optimal for $C_{\max} = 2$ but we now can decrease the makespan again by shortening p_1 and p_2 down to $1 = p_1^{min} = p_2^{min}$. With $p = (1, 1, 4)$ we achieve the smallest possible makespan $C_{\max} = 2$ at total cost $\kappa(p) = 24/5$. These observations are partially summarised in the following lemma.

Lemma 2.6.3 (Stork (2001)). *If AND/OR precedence constraints are imposed among jobs, then the time-cost trade-off curve $B(T)$ is non-increasing and piecewise linear. In general, it is not continuous (and thus not convex).*

We want to make an additional remark on this subject.

Remark 2.6.4. *In the time-cost trade-off problem with AND/OR precedence constraints among the jobs, in general there does not exist a continuous change of the processing time vector realising the time-cost trade-off curve $B(T)$, even if $B(T)$ itself is continuous. In addition, it is not optimal in general to decrease the processing times of the jobs in a cut-critical set.*

After this discussion of a specific example for the time-cost trade-off problem, it seems to be obvious that computing $B(T)$ is much harder in the presence of AND/OR precedence constraints than in the case of standard precedence constraints. In fact, the time-cost trade-off problem is strongly NP-hard for AND/OR-

networks. Already a special case of the problem, the so-called DEADLINE PROBLEM can be proved to be NP-complete.

> DEADLINE PROBLEM: For a given deadline $T \geq 0$ and an integer budget b, does there exist a processing time vector p with $C_{\max}(p) \leq T$ and $\kappa(p) \leq b$?

Theorem 2.6.5 (Stork (2001)). *The* DEADLINE PROBLEM *in AND/OR-networks is NP-complete in the strong sense.*

The original proof in Stork (2001) is done by a reduction from HITTING SET.

> HITTING SET: Let $U = \{x_1, \ldots, x_n\}$ be a finite ground set, $\mathcal{S} = \{S_1, \ldots, S_m\}$ a set of subsets of U, and k be a positive integer. Does there exist a subset $U' \subseteq U$ with $|U'| \leq k$ that contains at least one element from each set S_j?

We will follow this proof but use a reduction from SET COVER, as we have introduced this before in Section 2.3.3. Ausiello, d'Atri, and Protasi (1980) have shown that HITTING SET and SET COVER are equivalent and actually can be translated into each other in a straightforward manner. Both problems are well known to be strongly NP-complete, see Garey and Johnson (1979) for example.

Proof. A processing time vector p fulfilling the two conditions is a polynomially checkable certificate for the problem, thus the DEADLINE PROBLEM is in NP.

Let an instance of SET COVER be given by a ground set $U = \{x_1, \ldots, x_n\}$, a set $\{S_1, \ldots, S_m\}$ of subsets of U and an integer $k > 0$. Recall the reduction from SET COVER to scheduling with AND/OR-networks presented in the proof of Theorem 2.3.7. The constructed AND/OR-network $N = (V \cup W, E)$ looks as follows. The job set $V = \mathcal{S} \cup \mathcal{I}$ contains a set node $S_j \in \mathcal{S}$ for each of the sets and an index node $i \in \mathcal{I}$ for each element in U. $W = \mathcal{E}$ is the set of n waiting conditions of the form $(\mathcal{S}^i, i)$, where $\mathcal{S}^i := \{S_j \mid x_i \in S_j\}$. We refer to the vertices in $\mathcal{E}$ as element nodes. The resulting network for a specific instance of SET COVER is presented in Figure 2.4. All index nodes $i \in \mathcal{I}$ are assigned a constant unit processing time and do not cause any costs. The set nodes $S_j \in \mathcal{S}$ have $p_{S_j}^{min} = 1$ and $p_{S_j}^{max} = 2$ with cost function $\kappa_{S_j}(p_{S_j}) = -p_{S_j} + 2$ for all $j = 1, \ldots, m$. We now ask whether there exists a processing time vector p such that $C_{\max}(p) \leq 2$ and $\kappa(p) \leq k$. We will prove that this special instance of the DEADLINE PROBLEM is equivalent to the SET COVER problem.

Suppose the instance of the SET COVER problem is a 'yes' instance. Then there exists a subset $\mathcal{S}' \subseteq \mathcal{S}$ with $|\mathcal{S}'| \leq k$ and every element $x_i \in U$ is contained in at least one set $S_j \in \mathcal{S}'$. For our DEADLINE PROBLEM this means that there exists

a subset $\mathcal{S}' \subseteq \mathcal{S}$ of set nodes such that $\mathcal{S}'$ includes a predecessor for each element node $x_i \in W$. Thus if we schedule all the set jobs S_j in $\mathcal{S}'$ with $p_{S_j} = p_{S_j}^{min} = 1$, involving costs of at most $k \cdot \kappa_{S_j}(p_{S_j}^{min}) = k$, we can start every index node $i \in \mathcal{I}$ at time 1. Altogether the project will finish with $C_{\max} = 2$. On the other hand, if the DEADLINE PROBLEM is a 'yes' instance, all the index nodes have to start at time 1 to be finished at time 2. Thus, every element node x_i has to have one predecessor that has a processing time of 1. As every set node S_j that is scheduled at its minimal processing time involves a cost of one, we know that there can be at most k such set nodes that are scheduled with $p_{S_j} = 1$, as we have total cost of at most k. Due to the construction we can thus conclude that there exists a subset $\mathcal{S}' \subseteq \mathcal{S}$ of sets, with $|\mathcal{S}'| \leq k$ and such that every element $x_i \in U$ is contained in at least one of the sets in $\mathcal{S}'$. □

We have seen that the pure scheduling problem is easy to solve. We have then concentrated on analysing AND/OR-networks and the behaviour of the makespan when processing times change. In contrast to scheduling with standard precedence constraints, critical jobs can be defined in four different ways which are not equivalent in general. Critical jobs come in critical sets and it is easy to find some critical set. In contrast to this it is NP-complete to decide whether a specific job is critical as we have seen in Section 2.4.2. In addition we have proved several features of critical sets and jobs like their fixed start times and the amount of influence they have on the makespan. The presented results provide a useful tool for the analysis of projects with AND/OR precedence constraints with respect to the stability of the project duration under changes in the processing times.

CHAPTER 3

SCHEDULING WITH AND/OR PRECEDENCE AND MACHINE CONSTRAINTS

This chapter is devoted to scheduling jobs with AND/OR precedence and additional machine constraints. We have seen in Section 1.2.2 that minimising the makespan of a project with AND/OR precedence constraints is easy and can be done in polynomial time. The main reason for this is that there are as many machines as needed for processing the jobs. Now, we will restrict the machines available to either one or several identical parallel machines. Our objective will be minimising the makespan, the total completion time or the total weighted completion time. As most of the problems are already NP-complete in the presence of standard precedence constraints, they are also NP-complete when AND/OR precedence constraints are involved.

In addition to the theoretical challenge of these problems, the work of this chapter is motivated mainly by applications in assembly sequencing problems, see Goldwasser and Motwani (1997) and Goldwasser and Motwani (1999) for example. If the task is to disassemble a product to reach a certain component, the different possibilities how to reach the component can be modeled by AND/OR precedence constraints. The disassembling has to be performed on one assembly line and it should be completed as soon as possible. This problem can be modeled by $1|ao\text{-}prec|C_{\max}$ for example. Nevertheless the results of this chapter might also be useful for resource constrained project scheduling. Consider the case that a set of jobs has to be scheduled subject to standard precedence constraints and resource constraints on m identical parallel machines. Suppose the resource constraints can be represented by a set of minimal forbidden sets and the size of this set is polynomial in the number of jobs to be scheduled. Representing the machine constraints as minimal forbidden sets in general involves an exponential number of forbidden sets. Thus it might be more efficient to compute an approximate solution on a small input than to compute some solution for an exponential input.

The chapter will be divided into three sections. First we will introduce a broadly applicable algorithm and then consider in one section the makespan and in another section the total weighted completion time objective. In each of these sections we will discuss the one machine and the several machine case. We will

mention some of the results for standard precedence constraints, which motivated our approach for the AND/OR precedence constraints, and present already known results for AND/OR precedence constraints. A host of results has been published for the case of standard precedence constraints and any of the objective functions, as well as a number of survey papers. We do not aim to give a complete overview of all the work that has been done in this area but rather try to present different techniques used to find new algorithms or substantially improve previously known approximation bounds.

For the objective of minimising the makespan, the problem with AND/OR precedence constraints can be reduced to the problem with standard precedence constraints preserving the approximation guarantee. This successful approach inspired us to consider standard precedence constraints for the objective of minimising the total weighted completion time as well. Unfortunately, there is a major gap between the approximability of the total weighted completion time subject to standard precedence constraints and subject to AND/OR precedence constraints.

When comparing the solutions computed by the discussed algorithms, we will always denote an *optimal schedule* for a given problem by $S^* = (S_1^*, \ldots, S_n^*)$. As most of the problems we will consider are NP-complete, we are interested in algorithms that run in time polynomial in the input size and produce provably good solutions. A polynomial time algorithm is called a *ρ-approximation algorithm* or simply *ρ-approximation*, if its solution is always within a factor of ρ of the optimal solution. The approximation ratio ρ is also called the *performance guarantee* or the *performance ratio*. We refer to Hochbaum (1997) for an introduction. A family of polynomial time approximation algorithms with performance guarantee $1+\varepsilon$ for all fixed $\varepsilon > 0$ is called a *polynomial time approximation scheme*, or *PTAS* for short. If the PTAS is also polynomial in $1/\varepsilon$ it is called a *fully polynomial time approximation scheme* or *FPTAS*.

Before the actual discussion about the different scheduling problems, we want to present an easy algorithm, applicable in many situations and often a basis for more sophisticated algorithms.

3.1 List Scheduling

One of the most basic and intuitive approaches to tackle a given scheduling problem is *list scheduling*. The idea is to order the jobs according to some priority rule, which gives an ordered list. Whenever a machine is free, the first job in the list that has not been scheduled yet and is available at that time is assigned to be processed by the free machine. We need some notation to formalise this process. Let $N = (V \cup W, E)$ be a feasible AND/OR-network with strictly positive processing time vector $p > 0$ and a positive release time vector $r \geq 0$ (if we do not

Algorithm 3: List Scheduling

Input: feasible AND/OR-network $N = (V \cup W, E)$, number $m \geq 1$ of machines, processing time vector $p > 0$, release dates $r \geq 0$, and priority list L of the jobs in V

Output: schedule $S = (S_1, \ldots, S_n)$

```
set t' := 0;
while list L ≠ ∅ do
    let t ≥ t' be the minimal time a machine M gets free;
    if there exist a job in L that is available at time t then
        among all jobs available at time t, let j be the first one in L;
        assign j to machine M and remove j from L;
        set S_j := t, and t' := t;
    else
        let t' > t be the minimal time a job in L becomes available;
        among all jobs that become available at t', let j be the first in L;
        assign j to machine M and remove j from L;
        set S_j := t';
return S;
```

explicitly impose release dates, we assume $r = 0$). Let $U \subseteq V$ be some subset of jobs containing start vertex s. A feasible schedule $S = (S_s = 0, S_1, \ldots, S_n, S_t)$ with $S_j < \infty$ for $j \in U$ and $S_j = \infty$ for $j \in V \setminus U$, is called a *partial schedule*.

Definition 3.1.1. *Let V be a set of jobs with processing times p, release dates r and AND/OR precedence constraints. In a partial schedule a job $j \in V$ is called* available *at time t, if*

- $t \geq r_j$ *and*
- $S_i + p_i \leq t$ *for every* $i \in ImPred(j)$.

We refer to Uetz (2001) for a similar definition of a partial schedule and a short overview over list scheduling and a variant called *job-based list scheduling*. We want to present the list scheduling algorithm formally, but we will not go into implementational details. According to Lawler, Lenstra, Rinnooy Kan, and Shmoys (1993), this algorithm has been presented first by Graham (1966) and we will also refer to it as *Graham's list scheduling algorithm*.

Note that all time points t are increasing completion times or release dates of jobs. The list scheduling algorithm can be implemented to run in time polynomial in the size of the input. In his paper, Graham (1966) presents a worst-case analysis of his list scheduling algorithm for $P|prec|C_{\max}$.

It is also called a priority-driven heuristic, according to the priority list that guides the order in which the jobs are scheduled. In most of the algorithms we will consider in this chapter, list scheduling will be a basic component. Priority driven heuristics never intentionally leave machines idle. A machine is only left idle if there are currently no jobs available. This means that for every job in the list, at least one direct predecessor is still in process. Recursing this argument gives a well known fact for standard precedence constraints stated in the next lemma. Let $G = (V, E)$ be a standard (acyclic) precedence graph with a strictly positive processing time vector p and $u, v \in V$ two nodes such that there exists a u-v-path in G. As a digraph G is just a special case of an AND/OR-network, we use the same notation for G as for AND/OR-networks. Let $P_{uv} = u = u_0, u_1, \ldots, u_k = v$ denote a (directed) path from u to v like it was introduced in Section 1.1.1. Recall that the *length* of path P_{uv} is defined as $l_{uv} = \sum_{i=0}^{k} p_{u_i}$. The length of a longest path from u to v shall be denoted by $l_{uv}^{\max}$. Formally, the length of a longest u-v-path in G is defined by $l_{uv}^{\max} = p_v$ if $u = v$ and $l_{uv}^{\max} = p_v + \max\{l_{ux}^{\max} \mid (x, v) \in E$ and there exists a u-x-path in $G\}$ if $u \neq v$.

Lemma 3.1.2 (Graham (1966)). *In any list schedule for a set of precedence constrained jobs, there is an s-t-path of jobs that is executed during all periods when some machine is idle, and the length of this path is not longer than the makespan of an optimal schedule.*

The big advantage of Graham's list scheduling is that it is easy, both to understand and to implement, and that it is applicable for a wide range of problems. Unfortunately the schedule computed by Graham's list scheduling is neither monotone nor continuous as for example the earliest start schedule, see Lemma 1.2.4. This disadvantage is caused by the greedy approach of the algorithm. There exist instances of $P|prec|C_{\max}$ for which relaxing the precedence constraints, continuously decreasing the processing times of the jobs, or increasing the number of machines available can increase the objective function $C_{\max}$. These phenomena are known as *Graham anomalies*, presented in Graham (1966). Despite this shortcoming, list scheduling has been successfully applied to a number of problems.

3.2 The Makespan

The makespan is certainly one of the most relevant objective functions for many applications. No matter what aims are pursued in a real project, the overall completion time of the project in general plays an important role. Minimising the makespan of a set of precedence constrained jobs on one machine like an assembly line, a worker or one processor, is trivial. If the jobs are not restricted at all, we can simply schedule them in any order without interruption and idle time in between

and the project completion time is the sum of the processing times of all jobs. If there are standard precedence constraints involved, we have to schedule the jobs according to these restrictions. That can be done by processing them in order of a linear extension of the partial order represented by the precedence constraints. In the case of AND/OR precedence constraints the solution is equally simple, the jobs can be executed in the order of a linear realisation of the AND/OR-network representing the AND/OR precedence constraints. In any case, the makespan is equal to the sum of the processing times of the jobs, assuming that the constraints are feasible of course. The situation changes as soon as there are several machines available to process the jobs.

3.2.1 Scheduling on Identical Parallel Machines

We now consider the case of scheduling a set of jobs restricted by either standard or AND/OR precedence constraints on m identical parallel machines. The problem $Pm|\,|C_{\max}$ is already NP-complete for $m = 2$, but can be solved in pseudo-polynomial time for any fixed number m of machines. For an arbitrary number of machines, that is $P|\,|C_{\max}$, the problem is NP-complete in the strong sense, see Garey and Johnson (1979).

Nevertheless, Graham's list scheduling (GLS) performs provably well for this problem. Let S^{GLS} denote the schedule produced by list scheduling for $P|\,|C_{\max}$ on m machines. Graham (1966) proved that for any instance,

$$C_{\max}(S^{GLS}) \;\leq\; (2 - \frac{1}{m})\, C_{\max}(S^*),$$

and this performance guarantee is tight. The upper bound on Graham's list scheduling can be established as follows. Let k be the last job completed in S^{GLS}, then there cannot occur any idle time before $t = C_{\max}(S^{GLS}) - p_k$. For the makespan of the list schedule it follows that

$$C_{\max}(S^{GLS}) = t + p_k \leq \frac{1}{m} \sum_{j \in V \setminus \{k\}} p_j + p_k = \frac{1}{m} \sum_{j \in V} p_j + \frac{m-1}{m} p_k.$$

The makespan of any optimal schedule is bounded from below by $C_{\max}(S^*) \geq \frac{1}{m} \sum_{j \in V} p_j$ and $C_{\max}(S^*) \geq p_k$ which proves the desired result.

Up to now, we have not used the fact that we can choose the order in which the jobs are processed by list scheduling. The bound proved by Graham (1966) holds for any list but actually, we can do better. Some years later, Graham showed that it is advantageous to take long jobs first. For list scheduling with *longest processing time (LPT)* rule, which means that we perform list scheduling in order

of non-increasing processing times of the jobs, the bound can be improved to

$$C_{\max}(S^{LPT}) \leq (\frac{4}{3} - \frac{1}{3m})\, C_{\max}(S^*).$$

After the pioneering work of Graham, a variety of approaches to $P|\,|C_{\max}$, in part other than list scheduling, have been proposed with different performance guarantees. We refer to Lawler et al. (1993) for an overview and the corresponding references.

Ullman (1975) first showed that the problem $P|prec, p_j = 1|C_{\max}$ is NP-hard. Only three years later, Lenstra and Rinnooy Kan (1978) proved that it is even NP-complete to decide whether there exists a feasible schedule of length at most three for this problem. This fact implies that unless P=NP, there does not exist a ρ-approximation algorithm for any factor $\rho < 4/3$. The proof is performed by a reduction from the NP-complete CLIQUE problem and can be found in Lawler et al. (1993), for example.

The restricted problem $P2|prec, p_j = 1|C_{\max}$ can be solved in polynomial time. The algorithm presented by Fujii, Kasami, and Ninomiya (1969) computes an optimal schedule in polynomial time by constructing a maximum cardinality matching in the incomparability graph of the precedence graph. Another variant that is polynomially solvable is $P|tree, p_j = 1|C_{\max}$. Hu (1961) presents the first polynomial time algorithm for this special case of precedence constraints that have a tree structure. There is a number of publications on similar special cases, according references again can be found in Lawler et al. (1993).

We have already noted that, unless P=NP, there is no polynomial time approximation algorithm for $P|prec, p_j = 1|C_{\max}$ achieving a ratio better than $4/3$. Nevertheless, list scheduling again performs well even on the general problem $P|prec|C_{\max}$. Graham (1966) proved that the worst-case performance of his list scheduling algorithm is not affected by the precedence constraints and thus

$$C_{\max}(S^{GLS}) \leq (2 - \frac{1}{m})\, C_{\max}(S^*).$$

Again the performance guarantee is tight, even if a special, the so-called *critical path* rule is used for the list scheduling.

Let us finally come to the problem with AND/OR precedence constraints. $P|ao\text{-}prec|C_{\max}$ is a generalisation of $P|prec|C_{\max}$ and is thus NP-complete. Nevertheless, a 2-approximation algorithm has been presented by Gillies and Liu (1995) for the case that the precedence graph represented by the AND/OR-network is a partial order, that is, the AND/OR-network does not contain any cycles. In their paper, Gillies and Liu also consider a variant of this problem, where exactly one predecessor of every OR-node has to be scheduled and all the others are left unscheduled. They refer to this problem as the *AND/OR/skipped*

Algorithm 4: Minimum Path Heuristic

Input: feasible AND/OR-network $N = (V \cup W, E)$, number $m \geq 1$ of machines, processing time vector $p > 0$

Output: schedule $S = (S_1, \ldots, S_n)$

compute earliest start schedule ES for (N, p) (no machine constraints);
let $G = (V, E(G))$ be the subgraph of N induced by V;
foreach *OR-node* $w = (X, j) \in W$ **do**
  for *exactly one* $x \in X$ *with* $ES_x + p_x = ES_w$ **do**
    add edge (x, j) to $E(G)$;
let L be a linear extension of the resulting AND-graph;
apply List Scheduling to the AND-graph G with p and L;
return S;

model. It turns out that this problem is actually harder than the *unskipped* variant. They also prove several NP-hardness results for different variants of the skipped and unskipped case. We will not go into any further details and explicitly concentrate on the approximation algorithm presented for the *AND/OR/unskipped* model which corresponds to $P|ao\text{-}prec|C_{\max}$ for acyclic AND/OR-networks in our notation. The basic idea of the so-called *Minimum Path Heuristic* of Gillies and Liu is to first change the AND/OR-network into an AND-only network, that is a standard precedence digraph, and then take Graham's list scheduling to find a schedule. We present the Minimum Path Heuristic of Gillies and Liu (1995) extended to AND/OR-networks that may contain cycles. Therefore we need the earliest start schedule of Section 1.2.2. By construction, the earliest start schedule minimises the longest path from s to any vertex u without violating any AND/OR constraints.

The strategy of the Minimum Path Heuristic is to fix the predecessor of an OR-node to the one that minimises the longest path to that OR-node, thereby turning the AND/OR-network into a standard precedence graph G. This is done by taking one of the tight direct predecessors of an OR-node in the earliest start schedule and making it a direct predecessor of the immediate successor of the OR-node. Note that this precedence graph G is a realisation of N. Every step of the Minimum Path Heuristic (MPH) can be computed in polynomial time, thus the Minimum Path Heuristic can be implemented to run in polynomial time.

Together with Lemma 3.1.2 by Graham and the optimality of the earliest start schedule with respect to the makespan without machine constraints, we are able to prove the same performance guarantee as Gillies and Liu for the Minimum Path Heuristic.

Theorem 3.2.1. *Let S^{MPH} denote the schedule computed by the Minimum Path Heuristic and S^* an optimal schedule for an instance of $P|ao\text{-}prec|C_{\max}$, then it holds that*

$$C_{\max}(S^{MPH}) \leq (2 - \frac{1}{m})\, C_{\max}(S^*).$$

Moreover, this bound is tight.

Proof. Consider the schedule S^{MPH} and its makespan $C_{\max}(S^{MPH})$ computed by the Minimum Path Heuristic. At any time $0 < t \leq C_{\max}(S^{MPH})$ it holds that either all machines are busy or that one or more machines are idle. Accordingly, we can divide the time between 0 and $C_{\max}(S^{MPH})$ into busy periods and idle periods. We denote the total length of all busy periods by T_b and the total length of all idle periods by T_i, thus

$$C_{\max}(S^{MPH}) = T_b + T_i.$$

For the total length of idle periods it holds that $T_i \leq l_{st}^{\max}(G)$ by Lemma 3.1.2. From the construction of graph G we get that $l_{st}^{\max}(G) = C_{\max}(ES)$, where ES is the earliest start schedule of (N, p) without machine constraints computed in the first step of the algorithm. In Proposition 1.2.3 we have stated that the earliest start schedule achieves the optimal makespan for an instance (N, p). Thus for the optimal makespan $C_{\max}(S^*)$ of (N, p) on m machines it holds that $C_{\max}(ES) \leq C_{\max}(S^*)$. Together this yields

$$T_i \leq C_{\max}(S^*).$$

Additionally, we can compare the total processing time of all jobs with the optimal makespan and with the schedule computed by the Minimum Path Heuristic. A trivial bound on any feasible schedule - and thus also the optimal schedule - is $C_{\max}(S^*) \geq \frac{1}{m} \sum_{j \in V} p_j$. On the other hand, the Minimum Path Heuristic also has to process all the jobs. During a busy period, all m machines process some job, while during an idle period at least one machine processes some job. We therefore get that

$$mT_b + 1T_i \leq \sum_{j \in V} p_j \leq mC_{\max}(S^*).$$

The worst case for $C_{\max}(S^{MPH}) = T_b + T_i$ subject to the presented constraints is achieved for $T_i = C_{\max}(S^*)$ and $T_b = (1 - 1/m)C_{\max}(S^*)$, which yields the result.

Examples for standard precedence graphs that achieve the worst-case bound of Graham's list scheduling can be found in Graham (1966) or Gillies and Liu (1991). If we apply the Minimum Path Heuristic to such instances, there is nothing to be done in the first part of the algorithm and the performance guarantee will be

achieved by the second part, the list scheduling. This proves the tightness of the stated approximation ratio and completes the proof. □

We want to remark that we have modified the notation and algorithm presented by Gillies and Liu (1995) to fit our purpose. In the original paper, Gillies and Liu work on AND/OR precedence constrained jobs where the OR-nodes are normal jobs without any restrictions. As we have argued in Section 1.2, this does not collide with our structural assumptions about AND/OR-networks.

3.3 The Total Weighted Completion Time

For minimising the total weighted completion time of a set of jobs, a nearly uncountable number of publications can be found in the literature. The main motivation for considering this objective function comes from applications in compiler optimisation. Fast algorithms are needed to exploit the parallelism provided by pipelined, super-scalar, and very-long instruction word architectures. These issues have been addressed by Hennessy and Gross (1983) and Weiss and Smith (1987) for example. Chekuri, Johnson, Motwani, Natarajan, Rau, and Schlansker (1996) consider profile-driven code optimisation, where the total weighted completion time also appears as a relevant cost measure.

In contrast to the makespan objective, the total weighted completion time of a set of AND/OR constrained jobs is not approximable within any reasonable factor, as we will prove later. Therefore we can expect that the successful approaches for standard precedence constraints will not be applicable to AND/OR precedence constraints. Nevertheless we will give a short review of the methods and results that have been presented for minimising the total weighted completion time of precedence constrained jobs. For some of the approaches we will indicate why they fail for AND/OR precedence constraints.

3.3.1 Scheduling on One Machine

In contrast to the makespan objective, scheduling on one machine in general is not trivial for the total (weighted) completion time objective. Nevertheless, already Smith (1956) proposed a polynomial time algorithm for the scheduling problem without precedence constraints among the jobs, that is $1||\sum \omega_j C_j$. A simple exchange argument shows that Graham's list scheduling is optimal for $1||\sum \omega_j C_j$ if the jobs in the list are ordered in non-decreasing p_j/ω_j ratio. This job ordering rule is also known as *Smith's rule*. Of course, this result carries over to the problem of minimising the total completion time, $\sum C_j$, which is just a special case of minimising the total weighted completion time with $\omega_j = 1$ for all

$j \in V$. This simpler rule, where the jobs are scheduled in order of non-decreasing processing times p_j is known as the *shortest processing time (SPT)* rule.

As soon as precedence constraints are imposed among the jobs, both problems become NP-hard in the strong sense, as has been shown by Lawler (1978) for $1|prec|\sum \omega_j C_j$ and in the same year by Lenstra and Rinnooy Kan (1978) for the problem $1|prec|\sum C_j$.

Several approaches to approximate $1|prec|\sum \omega_j C_j$ have been presented by various research groups. The best known approximation algorithms achieve a performance guarantee of 2 and it is still an open problem to determine the exact approximation ratio of $1|prec|\sum \omega_j C_j$. Even for the special case of identical weights, no better performance guarantee was found. Woeginger (2001) presents a nice overview of the problem $1|prec|\sum \omega_j C_j$ and its special cases. He proved that altogether eight different versions of the problem have the same approximation ratio. We want to mention some of the interesting special cases.

Theorem 3.3.1 (Woeginger (2001)). *The approximation thresholds of the following special cases of the scheduling problem* $1|prec|\sum \omega_j C_j$ *all coincide:*

1. $1|prec|\sum \omega_j C_j$*, the general problem*
2. $1|prec|\sum C_j$*, the problem with identical weights*
3. $1|prec, p_j = 1|\sum \omega_j C_j$*, the problem with identical processing times*
4. $1|prec, \omega_j = 1, p_j \in \{0,1\}|\sum \omega_j C_j$
5. $1|prec, \omega_j \in \{0,1\}, p_j = 1|\sum \omega_j C_j$

The theorem shows that the problem of minimising the total weighted completion time with precedence constrained jobs is not easier to approximate even if the weights and the processing times are restricted to the values 0 and 1. In a subsequent paper of Kolliopoulos and Steiner (2002) about the partially-ordered knapsack problem, several other special cases of $1|prec|\sum \omega_j C_j$ are identified, for which there exist approximation algorithms with a performance guarantee of $1.62 + \varepsilon$.

It is an interesting fact that the exact approximation threshold for the total weighted completion time subject to precedence constraints is not known, but several essentially different approximation algorithms have been proposed, which all achieve an approximation guarantee of 2. Schulz (1996) and Hall, Schulz, Shmoys, and Wein (1997) present a 2-approximation for $1|prec|\sum \omega_j C_j$ that is based on a linear programming relaxation. The proposed algorithm is a list scheduling that uses the order of the completion times of an optimal solution to an LP-relaxation for the problem to compute a schedule. A main ingredient for

proving the performance ratio of 2 is the fact that the completion times of the jobs in an optimal solution for the LP-relaxation induce a linear extension of the precedence graph. In the case of AND/OR precedence constraints we can set up a similar integer program. However, when relaxing the integrality condition in this formulation, an optimal solution does not necessarily induce a linear realisation of the AND/OR-network. So we lose a crucial property for proving an approximation guarantee. Back to standard precedence constraints, a tighter analysis of the LP-relaxation algorithm is presented in Schulz (1996) that yields a $2-\frac{2}{n+1}$ approximation ratio, where n is the number of jobs. Schulz (1996) and Hall et al. (1997) also prove that essentially the same algorithm, where additional constraints for the release dates are included into the LP-relaxation, yields a performance guarantee of 3 for $1|r_j|\sum \omega_j C_j$ and $1|prec, r_j|\sum \omega_j C_j$. By now, better approximation algorithms have been proposed for the case with release dates as we will see further down.

Another approximation algorithm for the problem $1|prec|\sum \omega_j C_j$ by Chudak and Hochbaum (1999) uses a half integral linear programming relaxation and a minimum cut computation to achieve the performance guarantee of 2. A completely different approach has been presented independently of each other by Margot, Queyranne, and Wang (2000) and by Chekuri and Motwani (1999). Margot et al. as well as Chekuri and Motwani provide a very simple purely combinatorial 2-approximation algorithm. In this algorithm, a special combinatorial structure of the jobs which are scheduled first in an optimal solution is used to guide the procedure of the approximation algorithm. Although an equivalent combinatorial structure can be defined on AND/OR-networks, an optimal solution to the one machine problem with AND/OR precedence constraints in general does not have the required property. As we have already expected, successful strategies for standard precedence constraints resist any application in the case of AND/OR precedence constraints.

We want to mention that Horn (1972) proved that the problem is polynomially solvable when the precedence constraints have a tree-like structure, that is the problem $1|tree|\sum \omega_j C_j$. Polyhedral methods for scheduling have been used extensively to characterise polynomially solvable special cases as well as to compute optimal solutions. Queyranne and Schulz (1994) present a thorough survey of results in this area.

For the problem with release dates instead of precedence constraints, a performance ratio better than 2 can be proved. In Phillips, Stein, and Wein (1998) a simple 2-approximation for $1|r_j|\sum C_j$ is described that is based on converting an optimal preemptive schedule into a non-preemptive one. The preemptive problem, that is $1|r_j, pmtn|\sum C_j$, can be solved to optimality in polynomial time by list scheduling with *shortest remaining processing time* (SRPT) rule. Chekuri, Motwani, Natarajan, and Stein (2001) propose a $\frac{e}{e-1}$-approximation algorithm for

$1|r_j|\sum C_j$, that is based on list scheduling by so-called α-points. The key observation for the approximation ratio is that it is possible to define a distribution for α that yields the stated performance guarantee of $\frac{e}{e-1}$. For further details about and references for scheduling by α-points we refer to Skutella (2002). Goemans, Queyranne, Schulz, Skutella, and Wang (2002) proposed another α-points schedule achieving an approximation ratio of 1.69 for the problem with release dates and arbitrary weights, $1|r_j|\sum \omega_j C_j$. Both approximation algorithms have been outperformed by a polynomial time approximation scheme for $1|r_j|\sum \omega_j C_j$ presented in Afrati, Bampis, Chekuri, Karger, Kenyon, Khanna, Milis, Queyranne, Skutella, Stein, and Sviridenko (1999). For additional precedence constraints, that is the problem $1|r_j, prec|\sum \omega_j C_j$, Schulz and Skutella (1997) propose an $(e+\varepsilon)$-approximation, which is also based on scheduling with α-points.

Minimising the total weighted completion time of a set of jobs subject to AND/OR precedence constraints is much harder than subject to standard precedence constraints. In addition there appears a gap in the approximability between the unweighted and the weighted case, in contrast to the problem with standard precedence constraints. We will examine the performance of list scheduling with shortest processing time rule on the problem $1|ao\text{-}prec|\sum \omega_j C_j$ for both cases of unit and arbitrary weights. It turns out that the greedy approach of list scheduling with SPT rule results in a useful property of the computed schedule with respect to any other feasible schedule. This property will be a major ingredient for the proofs of a performance guarantees of $\sqrt{n}$ for the total completion time and a performance guarantees of n for the total weighted completion time of list scheduling with SPT rule. Let $(N = (V \cup W, E), p, \omega)$ be an instance of $1|ao\text{-}prec|\sum \omega_j C_j$ and S^{SPT} with completion time vector C^{SPT} the schedule computed for this instance by list scheduling with SPT rule. Consider an arbitrary feasible schedule S and its completion time vector C for (N, p, ω). For any job $j \in V$ we define a threshold $\xi^j(S)$ for j with respect to the feasible schedule S: $\xi^j(S) = \max\{\, p_i \mid C_i \leq C_j \,\}$. This threshold is the maximum processing time of a job equal to j or scheduled before j in S.

Lemma 3.3.2. *Let C^{SPT} be the completion time vector of a schedule computed by list scheduling according to shortest processing time rule for an instance (N, p, ω) and S be an arbitrary feasible schedule with thresholds $\xi^j(S)$ for all $j \in V$. Then for every job $j \in V$ it holds that $p_i \leq \xi^j(S)$ for all i with $C_i^{SPT} \leq C_j^{SPT}$.*

Proof. The proof is accomplished by induction over the order of the jobs in the feasible schedule S. Therefore we assume without loss of generality that the jobs are numbered such that $S_1 < S_2 < \cdots < S_n$. In addition, we write ξ for short instead of $\xi(S)$.

Consider job 1: By definition we get that $\xi^1 = p_1$. In addition we know that job 1 is available at time $t = 0$ as it is scheduled at that time in the feasible schedule S and thus cannot have any direct predecessors except s. By the greedy choice of list scheduling it follows that no job with a strictly longer processing time (and therefore coming after 1 in the list) will be scheduled before 1 in S^{SPT}.

Now consider job k and assume that for all $j = 1, \ldots, k-1$ it holds that $p_i \leq \xi^j$ for all i with $C_i^{SPT} \leq C_j^{SPT}$. We have to distinguish between the two cases that the threshold of k is greater or equal to the threshold of $k-1$.
Case a) $\xi^k > \xi^{k-1}$. From the definition of ξ it follows that $\xi^k = p_k$ and thus $p_k > \xi^{k-1}$. Let $x \in \{1, \ldots, k-1\}$ be the job that finishes last in the list schedule, that is $C_x^{SPT} = \max\{C_j^{SPT} \mid j = 1, \ldots, k-1\}$. By assumption it holds that $p_i \leq \xi^x \leq \xi^{k-1} < p_k$ for all i with $C_i^{SPT} \leq C_x^{SPT}$. We conclude that k is scheduled after x in the list schedule and $p_i \leq \xi^k$ for all i with $C_i^{SPT} \leq C_x^{SPT}$. From the feasibility of schedule S and the maximal choice of x in S^{SPT} it follows that k is available at time C_x^{SPT}. By the greedy choice of list scheduling, no job with a strictly longer processing time than k (and therefore coming after k in the list) will be scheduled after C_x^{SPT} and before C_k^{SPT}, which proves the first case.
Case b) $\xi^k = \xi^{k-1}$. This time we cannot say anything about the position of k in the schedule S^{SPT} and we have to distinguish between the cases that k finishes before or after some other job $j < k$. Again let $x \in \{1, \ldots, k-1\}$ be such that $C_x^{SPT} = \max\{C_j^{SPT} \mid j = 1, \ldots, k-1\}$. If k completes before x, that is $C_k^{SPT} < C_x^{SPT}$ then the claim trivially holds as by assumption $p_i \leq \xi^x \leq \xi^{k-1} = \xi^k$ for all i with $C_i^{SPT} \leq C_x^{SPT}$. If on the other hand k completes after x, we can again argue like in case a). From the feasibility of schedule S and the maximal choice of x in the list schedule S^{SPT} it follows that k is available at time C_x^{SPT}. By the greedy choice of list scheduling, no job with a strictly longer processing time than k will be scheduled after C_x^{SPT} and before C_k^{SPT}. Together with the property that the claim ($p_i \leq \xi^k$) trivially holds for all jobs i that are scheduled before x in S^{SPT} this completes the proof of the second case. □

Note that this property of the solution for an instance (N, p) computed by list scheduling with SPT rule holds for the threshold of any feasible schedule S and thus in particular for the threshold of an optimal solution S^*, independent of the optimality criteria. We can define a global threshold ξ^j for every vertex j of (N, p) by taking the minimum of the $\xi^j(S)$ over all feasible schedules S for (N, p). As we do not need it in the following, we omit any further details.

With this lemma we are able to prove two approximation guarantees of list scheduling with SPT rule for the problem of minimising the total completion time of AND/OR precedence constrained jobs on one machine. First we want to consider the special case of unit weights and prove that list scheduling with shortest processing time rule is a $\sqrt{n}$-approximation for $1|ao\text{-}prec|\sum C_j$, where n is the

number of jobs.

Theorem 3.3.3. *Scheduling a set of n AND/OR precedence constrained jobs in order of non-decreasing processing times (SPT rule) is a $\sqrt{n}$-approximation for the problem $1|ao\text{-}prec|\sum C_j$. Moreover, this bound is tight.*

Proof. Consider the schedule S^{SPT} with completion time vector C^{SPT} for an instance (N, p) of $1|ao\text{-}prec|\sum C_j$ with n jobs. Let $x \in V$ be the last job in the list schedule for which holds that x itself and every job completed before x in the list schedule have a processing time smaller or equal to $C_x^{SPT}/\sqrt{n}$, if such a job exists. Formally, x is chosen such that

$$C_x^{SPT} = \max_{j \in V}\{C_j^{SPT} \mid p_i \leq \frac{C_j^{SPT}}{\sqrt{n}} \; \forall i \text{ with } C_i^{SPT} \leq C_j^{SPT}\}.$$

If such an x exists, let $V^{\leq} = \{j \in V \mid C_j^{SPT} \leq C_x^{SPT}\}$ denote the set of jobs that are scheduled before x including x itself. If no such x exists, then $V^{\leq} = \emptyset$. The set of jobs scheduled after x in the list schedule shall be denoted by $V^{>} = \{j \in V \mid C_j^{SPT} > C_x^{SPT}\}$, where $V^{>} = V$, if no such x exists. We have to treat the jobs in the two sets separately.

First consider $V^{\leq}$. The total completion time of the list schedule for the jobs in $V^{\leq}$ can be generously estimated by

$$\sum_{j \in V^{\leq}} C_j^{SPT} \leq nC_x^{SPT}.$$

Now we have to bound the total completion time of an optimal schedule for the jobs in $V^{\leq}$. By construction, for every job $j \in V^{\leq}$ it holds that $p_j \leq C_x^{SPT}/\sqrt{n}$. Thus, there are at least $r \geq \sqrt{n}$ jobs in $V^{\leq}$ and we want to minimise their total completion time. Let $j_1, \ldots, j_r$ be the jobs in $V^{\leq}$ such that $C_{j_1}^* < \cdots < C_{j_r}^*$ and consider the processing times p_{j_i} of the jobs j_i, $i = 1, \ldots, r$ as variables. Then bounding the total completion time of the optimal schedule from below can be written as a linear program: Minimise $\sum_{i=1}^{r} C_{j_i}^*$ subject to the side constraints $0 < p_{j_i} \leq C_x^{SPT}/\sqrt{n}$ for all $i = 1, \ldots, r$ and $\sum_{i=1}^{r} p_{j_i} \geq C_x^{SPT}$. We get the following inequality, where we have turned around the addends:

$$\sum_{j \in V^{\leq}} C_j^* \geq C_x^{SPT} + (C_x^{SPT} - p_{j_r}) + (C_x^{SPT} - p_{j_r} - p_{j_{r-1}}) + \cdots + p_1$$

The sum on the right hand side is minimised if we descend from the maximal value C_x^{SPT} as fast as possible and in as few steps as possible. This is achieved if the processing time variable of every job j_i, $i = 1, \ldots, r$, takes its maximum

possible value $p_{j_i} = C_x^{SPT}/\sqrt{n}$. We then get

$$\sum_{j \in V^{\leq}} C_j^* \geq \sum_{i=1}^{\sqrt{n}} i \, \frac{C_x^{SPT}}{\sqrt{n}} \geq \frac{\sqrt{n}}{2} C_x^{SPT}$$

Bringing the two estimates together yields the desired approximation ratio for the jobs in $V^{\leq}$.

Now consider the jobs in $V^{>}$. By definition, for every $j \in V^{>}$ there exists a job k with $p_k > C_j^{SPT}/\sqrt{n}$ that is either equal to j or scheduled before j in the list schedule. Consider an optimal schedule S^* and its corresponding thresholds $\xi^j(S^*)$, that is the maximum processing time of a job scheduled before or equal to j in S^* as it was defined above. By Lemma 3.3.2 we know that every job scheduled before or equal to j in the list schedule has a processing time smaller than or equal to the threshold $\xi^j(S^*)$ of j in the optimal schedule. This yields

$$\frac{C_j^{SPT}}{\sqrt{n}} \; < \; p_k \; \leq \; \xi^j(S^*) \; \leq \; C_j^*.$$

Note that $\xi^j(S^*)$ is a trivial lower bound on C_j^* by definition. This is all we need to prove the approximation ratio stated in the theorem, as we can now estimate:

$$\begin{aligned} \sum_{j \in V} C_j^{SPT} &\leq \sum_{j \in V^{\leq}} C_j^{SPT} + \sum_{j \in V^{>}} C_j^{SPT} \\ &\leq 2\sqrt{n} \sum_{j \in V^{\leq}} C_j^* + \sum_{j \in V^{>}} \sqrt{n} C_j^* \leq 2\sqrt{n} \sum_{j \in V} C_j^* \end{aligned}$$

We have proved that list scheduling with SPT rule is a ρ-approximation with $\rho \in O(\sqrt{n})$, it remains to prove that also $\rho \in \Omega(\sqrt{n})$. Already for standard precedence graphs, the approximation ratio for list scheduling with SPT rule is bounded from below by $\sqrt{n}$ and therefore this also holds for the general case of AND/OR precedence constraints. We will present a series of digraphs that force the approximation ratio of the SPT rule a factor of $\sqrt{n}$ away from the optimum.

Example 3.3.4. *Let $k > 0$ be an integer and consider the following precedence graph $G = (V, E)$. The set of jobs V consists of k jobs $i_1, \ldots, i_k$, one job x, and k^2 jobs $j_1, \ldots, j_{k^2}$ that form a chain, preceded by x. Thus in E there are the edges (x, j_1) and $(j_\kappa, j_{\kappa+1})$ for all $1 \leq \kappa \leq k^2 - 1$. We have the following processing times on the jobs: $p_{i_\kappa} = k^2$, for $\kappa = 1, \ldots, k$, $p_x = k^2$, and $p_{j_\kappa} = 1$, for all $\kappa = 1, \ldots, k^2$.*

The SPT rule orders the jobs according to non-decreasing processing time and thus $L = j_1, \ldots, j_{k^2}, i_1, \ldots, i_k, x$, for example. The list scheduling then computes

a schedule with the following order of the jobs:

$$SPT: \boxed{i_1}\boxed{\ldots}\boxed{i_k}\boxed{x}\boxed{j_1}\boxed{\ldots}\boxed{j_{k^2}}$$

For the objective value of this schedule we get that

$$\sum_{j \in V} C_j^{SPT} \geq \sum_{\ell=1}^{k+1} \ell k^2 + k^2(k+1)k^2 = \Omega(k^5).$$

The optimal schedule prefers job x, which can release the long chain of short jobs. Therefore, an optimal solver produces the following schedule:

$$OPT: \boxed{x}\boxed{j_1}\boxed{\ldots}\boxed{j_{k^2}}\boxed{i_1}\boxed{\ldots}\boxed{i_k}$$

For the value of an optimal schedule we can calculate that

$$\sum_{j \in V} C_j^* \leq (k^2+1)2k^2 + \sum_{\ell=3}^{k+2} \ell k^2 = O(k^4)$$

Thus the performance ratio of list scheduling for this instance is $\Omega(k^5)/O(k^4) = \Omega(k)$. The precedence graph has $n \in \Theta(k^2)$ many vertices, which yields an approximation guarantee of $\Theta(\sqrt{n})$ for list scheduling with SPT rule on this instance for n respectively k going towards infinity. □

Coming back to the weighted problem $1|ao\text{-}prec|\sum \omega_j C_j$, we are actually able to prove a strong inapproximability result. Goldwasser and Motwani (1997) consider a special assembly sequencing problem: removing a given disk from a collection of unit disks in the plane. Goldwasser and Motwani prove that approximating the number of steps required to within a factor of $2^{\log^{1-\gamma} n}$ for any $\gamma > 0$ is quasi NP-hard. A problem is called *quasi NP-hard* if a polynomial time algorithm could be used to solve all NP-hard problems in *quasi polynomial* time, that is in $O(n^{poly(\log n)})$ time in contrast to $O(n^{O(1)})$ in the case of standard NP-hardness. For further references on approximability see Arora and Lund (1997) or Ausiello et al. (1999).

To establish the inapproximability result for the assembly problem, Goldwasser and Motwani (1997) show that this sequencing problem is a special case of scheduling with AND/OR precedence constraints. We will show that the scheduling problem with AND/OR precedence constraints considered in Goldwasser and Motwani is again a special case of $1|ao\text{-}prec|\sum \omega_j C_j$ thus proving the same inapproximability result for minimising the total weighted completion time subject to AND/OR precedence constraints. We first present the proof of Goldwasser and Motwani and explain afterwards how this relates to our problem.

Goldwasser and Motwani consider the problem of scheduling a set of jobs, which they call *tasks*. Each job has unit processing time and there are no weights involved. The jobs are either AND- or OR-nodes and Goldwasser and Motwani assume that the graph representing the precedence constraints has *internal-tree* structure. An internal-tree is a slight generalisation of a tree. We call a node of the network a *leaf*, if it has no direct predecessors. An AND/OR-network $N = (V \cup W, E)$ has internal-tree structure, if $N \setminus \{j \in V \mid j \text{ is a leaf}\}$ is an in-tree. The scheduling problem considered by Goldwasser and Motwani corresponds to the AND/OR/skipped model of Gillies and Liu (1995). This scheduling model requires that for an AND-node to be scheduled, all its direct predecessors have to be processed before the AND-node, and for an OR-node only one direct predecessor has to be scheduled before the OR-node and the others may be left completely unprocessed. The goal of AND/OR scheduling is to minimise the number of leaves or the number of jobs (AND- and OR-nodes) that need to be scheduled to be able to schedule some sink y of the network, and thus solving the problem instance. The next theorem tells us that this is a hard problem.

Theorem 3.3.5 (Goldwasser and Motwani (1997)). *It is quasi-NP-hard to find a solution which is within a factor of $2^{\log^{1-\gamma} n}$, for any $\gamma > 0$, of the optimal solution for any of the problems,*

a) minimising the number of leaves scheduled for AND/OR scheduling with internal-tree precedence constraints and

b) minimising the number of nodes scheduled in an AND/OR-network,

where $n = |V \cup W|$ is the number of nodes.

To prove the inapproximability of these problems, Goldwasser and Motwani show that LABEL COVER is a special case of problem a) and then prove that a) can be reduced to b). Goldwasser and Motwani use the definition of LABEL COVER presented in Arora and Lund (1997), which is a slightly restricted version. We will present the original definition of LABEL COVER that was introduced by Arora, Babai, Stern, and Sweedyk (1993). A final version of this paper has been published in Arora et al. (1997). LABEL COVER has been introduced in approximability theory as an artificial generalisation of the SET COVER problem. In Arora and Lund (1997) it is presented as one of six "canonical" problems for proving hardness of approximation.

> LABEL COVER: The input to the LABEL COVER problem is a bipartite graph $G = (V_1, V_2, E)$, two finite sets of labels B_1 and B_2 for the nodes in V_1 and V_2 respectively, and a relation $\Lambda \subseteq E \times B_1 \times B_2$, that consists of admissible pairs of labels for each edge $e \in E$. A *labelling*

of the graph is a pair of functions (f_1, f_2), where $f_i : V_i \to 2^{B_i}$ for $i = 1, 2$, that is a labelling assigns a (possibly empty) set of labels to each vertex of the graph. The *cost* of a labelling is $\sum_{v \in V_1} |f_1(v)|$. A labelling is said to *cover* an edge $e = (v_1, v_2)$, if both $f_1(v_1)$ and $f_2(v_2)$ are non-empty and for every label $b_2 \in f_2(v_2)$ assigned to v_2, there exists a label $b_1 \in f_1(v_1)$ such that $(e, b_1, b_2) \in \Lambda$. A *total cover* is a labelling that covers every edge. The goal is to find a total cover of minimum costs.

This minimisation problem of LABEL COVER has been proved by Arora et al. (1997) to be quasi-NP-hard to approximate within a factor of $2^{\log^{1-\gamma} n}$ of the optimum for any $\gamma > 0$, where $n = |V_1|$. We will now present the proof of Theorem 3.3.5 adapted to the presented definition of LABEL COVER.

Proof. a) Given an instance of the LABEL COVER problem, we will construct an AND/OR-network with internal-tree structure such that a solution to the scheduling problem that minimises the number of scheduled leaves corresponds one-to-one to a minimum LABEL COVER. The AND/OR-network $N = (V \cup W, E)$ has five layers of nodes, alternating between AND- and OR-nodes.

Let a hard instance of LABEL COVER be given by $G = (V_1, V_2, E)$, B_1 and B_2, and $\Lambda \subseteq E \times B_1 \times B_2$ as specified above. The inapproximability factor of LABEL COVER depends only on the size of V_2, as we will see in Theorem 3.3.6. Thus we can assume that $n = |V_2|$ and $|V_1|$, $|B_1|$, and $|B_2|$ are polynomially bounded in n.

The construction is backward, that is we start with the last nodes. The source of the network is again represented by the dummy node s, which precedes all leaves, and the last vertex is t, which represents the completion of the problem. The levels are constructed as follows:

- The first level contains one AND-node u, which precedes t.
- The second level contains an OR-node for every node $v_2 \in V_2$. Each of these OR-nodes has AND-node u as direct successor. Therefore, these nodes ensure that every node in V_2 gets assigned at least one label.
- The third level has an AND-node for each pair $\langle v_2, b_2 \rangle$, where $v_2 \in V_2$ and $b_2 \in B_2$, which is a direct predecessor of OR-node v_2 of level two. These nodes represent the possible labelling of a node $v_2 \in V_2$ and guarantee that for a label b_2 assigned to v_2, it must be the case that for every edge $e = (v_1, v_2)$ incident to v_2, the function f_1 respects the labelling.
- The fourth level contains an OR-node $\langle e, b_2 \rangle$, where $e = (v_1, v_2)$ is an edge incident to v_2 and $b_2 \in B_2$. The OR-node $\langle e, b_2 \rangle$ is a direct predecessor of $\langle v_2, b_2 \rangle$ of the third level. It ensures that if label b_2 is assigned to v_2, then

edge e can only be covered if some b_1 is assigned to v_1 such that $(e, b_1, b_2) \in \Lambda$.

- The fifth and last level contains the leaf nodes. There is one leaf for every pair $\langle v_1, b_1 \rangle$ representing that label b_1 is assigned to v_1. A leaf $\langle v_1, b_1 \rangle$ is a direct predecessor of OR-node $\langle e, b_2 \rangle$ of level four, with $e = (v_1, v_2)$, if $(e, b_1, b_2) \in \Lambda$.

By construction, the AND/OR-network has internal-tree structure. In addition there is a one-to-one correspondence between a valid labelling and a feasible solution to the AND/OR scheduling instance. It is easy to see that a minimum LABEL COVER corresponds to a solution to the AND/OR scheduling instance that minimises the number of scheduled leaves and vice versa. The size of the AND/OR scheduling instance is polynomially bounded in n by construction. This yields that the inapproximability result of LABEL COVER carries over to the problem of scheduling a minimum number of leaves of AND/OR precedence constrained jobs with internal-tree precedence structure.

b) For this problem, we show how to reduce the problem of minimising the number of scheduled leaves of an AND/OR-network with internal-tree structure to the problem of minimising the number of jobs scheduled subject to arbitrary AND/OR precedence constraints. The difficulty in the reduction is that the number of internal nodes might be significant and thus we could not guarantee for the same performance ratio. This problem can be overcome by increasing the costs of scheduling leaves. Consider a hard instance of the leaf minimisation problem constructed in a) with i internal nodes and ℓ leaves. We convert this instance into an instance with general AND/OR-network by replacing each leaf by a chain of αi jobs, for some constant $\alpha > 0$. The new instance has $i + \alpha i\ell$ nodes. The hard instance of LABEL COVER used in our construction fulfills that $|V_2| = n$ and $|V_1|$, $|B_1|$, and $|B_2|$ are polynomially bounded in n. From this and the construction in the proof of a) it follows that the number i of internal nodes is polynomially bounded in ℓ, the number of leaves. Thus, $i + \alpha i\ell$ is also polynomially bounded in ℓ. With the right choice of α, the additive error in the performance guarantee can be made arbitrarily small which yields the desired result: Let $i + \alpha i\ell = n' = poly(\ell)$, then $2^{\log^{1-\gamma} n'}$ is less than $2^{\log^{1-\gamma'} n}$ for an appropriate $\gamma' > 0$. This concludes the proof. □

Goldwasser and Motwani (1997) use this theorem to prove the inapproximability of reaching a certain disk out of a collection of disks in the plane, where only translations to infinity into two directions are allowed. The remarkable fact about this result is that it can be realised geometrically in the plane on a grid, polynomially bounded in the input size. The result can also be extended to axis-aligned unit squares as well as to higher dimensions.

We want to mention that the scheduling problem of minimising the number of leaves scheduled is similar to the AND/OR/skipped model of Gillies and Liu (1995), where predecessors of OR-tasks are allowed to remain unscheduled. The main motivation for this kind of scheduling stems from maintenance and recycling problems. A classical example of a maintenance problem is that we have to exchange a broken part in some product, let's say a plug in a car, without taking the whole product apart. In recycling applications the goal is to disassemble an old product to reach a valuable part with minimal effort. Goldwasser and Motwani (1999) present a comprehensive overview over assembly sequencing problems. They classify them into different groups according to the possible goal, such as full disassembly or remove a key part for example, the possible restrictions, such as linear sequencing or constant number of possible motions, and possible measures, like smallest number of steps or fewest re-orientations for example. Then Gillies and Liu present approximation preserving reductions between the different variants of assembly sequencing thereby proving inapproximability results for a variety of assembly problems. As a stepping stone, Goldwasser and Motwani again consider the problem of AND/OR scheduling.

Now we will explain, how the result of Goldwasser and Motwani (1997) carries over to the scheduling problem $1|ao\text{-}prec|\sum \omega_j C_j$. Dinur and Safra (1999) strengthen the inapproximability result of Arora et al. (1997) in two ways. On one side they provide an NP-hardness result opposed to the quasi-NP-hardness of Arora et al. and on the other side they increase the factor of non-approximability of LABEL COVER. This stronger inapproximability result carries over to the problem of minimising the total weighted completion time of AND/OR precedence constrained jobs on one machine. Let LABEL COVER be given by a bipartite graph $G = (V_1, V_2, E)$, two finite sets of labels B_1 and B_2 and a relation $\Lambda \subseteq E \times B_1 \times B_2$ as described above.

Theorem 3.3.6 (Dinur and Safra (1999)). *It is NP-hard to approximate the costs $\sum_{v \in V_1} |f_1(v)|$ of a minimum total cover to within a factor of $2^{\log^{1-\gamma} n}$, where $\gamma = 1/(\log\log n)^c$ for any constant $c < 1/2$ and $n = |V_2|$.*

Now we can formulate the inapproximability result for $1|ao\text{-}prec|\sum \omega_j C_j$. We can restrict to the special case of unit processing times, as we will see.

Theorem 3.3.7. *The scheduling problem $1|ao\text{-}prec, p_j = 1|\sum \omega_j C_j$ is NP-hard to approximate within a factor of $2^{\log^{1-\gamma} n}$ times the optimum value, where $\gamma = 1/(\log\log n)^c$ for any constant $c < 1/2$ and $n = |V|$ is the number of jobs.*

Proof. The proof is done by successively applying the theorems we have just presented. We start with the inapproximability result of LABEL COVER. By Theorem 3.3.6 we know that LABEL COVER is NP-hard to approximate within

a factor of $2^{\log^{1-1/\log\log^c n} n}$ times the optimum, for any constant $c < 1/2$. Now we apply the construction used in the proof of Theorem 3.3.5 to get an instance of AND/OR-constrained jobs with internal-tree structure and the same inapproximability bound. By Theorem 3.3.5 case b) we know that this NP-hardness result can be extended to the problem of minimising the scheduled jobs in an AND/OR-network, maintaining the inapproximability factor. Consider the constructed network $N = (V \cup W, E)$. We assign unit processing times to all AND-nodes, thus in particular to the leaf nodes, respectively the chains they have been replaced by. Note that we even lose some of the overhead of internal nodes as there is only one layer of internal nodes that consists of AND-nodes thus having a positive processing time. In addition, all jobs except u have zero weight, $w_j = 0$ for all $j \in V \setminus \{u\}$. The node u has weight $w_u = 1$. The solution S' to the problem of scheduling a minimum number of AND/OR constrained jobs induces a solution to $1|ao\text{-}prec, p_j = 1|\sum \omega_j C_j$ that minimises the total weighted completion time: Simply schedule all jobs contained in S' before u and all other other jobs afterwards. The total weighted completion time is equal to the number of AND-nodes of solution S', thus in particular it contains the number of scheduled leaves. By the same argumentation as in the proof of Theorem 3.3.5, b) we get the desired inapproximability bound. Note that by construction, the number of internal AND-nodes as well as the number of leaves are polynomially bounded in n, thus $|V|$ is polynomial in n, which preserves the inapproximability bound. □

Theorem 3.3.7 shows that $1|ao\text{-}prec|\sum \omega_j C_j$ is a very hard problem even if all processing times are equal to one. Interestingly enough, the next theorem proves that list scheduling with shortest processing time rule is an easy n-approximation for this problem and we consider it unlikely that there exists an approximation algorithm achieving an asymptotically better ratio.

Theorem 3.3.8. *Scheduling a set of n weighted AND/OR precedence constrained jobs in order of non-decreasing processing times (SPT rule) is an n-approximation for the problem $1|ao\text{-}prec|\sum \omega_j C_j$. Moreover, this bound is tight.*

Proof. Consider an AND/OR-network $N = (V \cup W, E)$ on n jobs with processing time vector p and weights ω. Without loss of generality assume that the jobs are numbered such that $p_1 \leq p_2 \leq \cdots \leq p_n$, that is in non-decreasing order of their processing times. Let S^{SPT} and C^{SPT} denote the schedule and its completion time vector produced by Graham's list schedule with the list $L = 1, \ldots, n$. The optimal schedule and its completion time vector are denoted by S^* and C^* as usual. We again need the threshold value $\xi^j(S^*) = \max\{p_i \mid C_i^* \leq C_j^*\}$ for a job $j \in V$, as we defined it for Lemma 3.3.2. This gives us a trivial lower bound on the completion time of each job $j \in V$ in the optimal schedule:

$$C_j^* \geq \xi^j(S^*)$$

Now we have to bound the completion time of j in the list schedule with SPT rule. Lemma 3.3.2 states that for each $j \in V$ it holds that $p_i \leq \xi^j(S^*)$ for all jobs i with $C_i^{SPT} \leq C_j^{SPT}$. Therefore we get the following upper bound on the completion time of j in the list schedule:

$$C_j^{SPT} \leq \sum_{p_i \leq \xi^j(S^*)} p_i.$$

Now we can simply put the two bounds together to prove the theorem. For every $j \in V$ it holds that

$$C_j^{SPT} \leq \sum_{p_i \leq \xi^j(S^*)} p_i \leq \sum_{p_i \leq \xi^j(S^*)} C_j^* \leq nC_j^*$$

and thus $\sum_{j \in V} \omega_j C_j^{SPT} \leq n \sum_{j \in V} \omega_j C_j^*$, which concludes the proof of the upper bound on the approximation ratio.

To prove the lower bound on the approximation ratio of list scheduling with SPT rule for the problem $1|ao\text{-}prec|\sum \omega_j C_j$, we present a series of AND/OR-networks that force the solution of list scheduling a factor of n away from the optimum.

Example 3.3.9. *Let $k > 0$ be an integer and consider the following AND/OR-network $N = (V \cup W, E)$. The set of jobs V consists of $k+2$ jobs, $i_1, \ldots, i_k$, x, and j. In addition there is one OR-node $w \in W$. The edge set E contains edges $(i_\kappa, i_{\kappa+1})$ for $\kappa = 1, \ldots, k-1$, (i_k, w), (x, w), and (w, j). The processing times of all jobs are equal to one. The weights are as follows, $\omega_{i_\kappa} = 0$ for all $\kappa = 1, \ldots, k$, $\omega_x = 0$, and $\omega_j = 1$.*

The constructed AND/OR-network is very simple, it consists of one OR-node with its direct successor. The two predecessors of the OR-node are one chain of k nodes and one single node. If we sort the jobs in order of non-decreasing processing times, we get the list $L = j, i_1, \ldots, i_k, x$, for example. The list scheduling then computes a schedule with the following order of the jobs:

$$SPT: \begin{array}{|c|c|c|c|c|} \hline i_1 & \ldots & i_k & j & x \\ \hline \end{array}$$

For the objective value of this schedule we get that

$$\sum_{j \in V} \omega_j C_j^{SPT} = k + 1.$$

The optimal schedule prefers job x, as it can release j, which is the only job with positive weight. Therefore, an optimal solver produces the following schedule:

$$OPT: \begin{array}{|c|c|c|c|c|} \hline x & j & i_1 & \ldots & i_k \\ \hline \end{array}$$

The value of an optimal schedule is

$$\sum_{j \in V} \omega_j C_j^* = 2.$$

Thus the performance ratio of list scheduling for this instance is $\frac{k+1}{2} = \Theta(k)$. The AND/OR-network has $n = k + 3 \in \Theta(k)$ many vertices, which yields a performance ratio of $\Theta(n)$ for the list scheduling algorithm on this instance for n respectively k going towards infinity. This establishes the lower bound for the approximation guarantee of the shortest processing time rule for the problem $1|ao\text{-}prec|\sum \omega_j C_j$. □

The constructed example has very restricted values, thus the performance guarantee even holds for the problem $1|ao\text{-}prec, p_j = 1, \omega_j \in \{0,1\}|\sum \omega_j C_j$. In addition, it also provides a lower bound of n for list scheduling with Smith's rule.

We want to briefly summarise the results stated in this section. Theorem 3.3.1 tells us that the approximation guarantee for minimising the total weighted completion time of precedence constrained jobs is the same for the general problem as well as for a number of special cases including unit processing times, unit weights, and combinations thereof. For AND/OR precedence constrained jobs Theorem 3.3.7 states that the general case $1|ao\text{-}prec|\sum \omega_j C_j$, is close to being not approximable within a factor of n, whereas according to Theorem 3.3.3 there exists a $\sqrt{n}$-approximation algorithm for the unweighted case. Thus it might be possible that there is a major gap between the hardness of approximating the total weighted and the total unweighted completion time subject to AND/OR precedence constraints. We also want to remark that the SPT rule might not be a good approach for the total completion time as the lower bound on its performance guarantee can already be established for standard precedence constraints. We also could not prove any inapproximability result for $1|ao\text{-}prec|\sum C_j$. It is still possible that there exists a constant or at least logarithmic approximation algorithm for the problem $1|ao\text{-}prec|\sum C_j$. For $1|ao\text{-}prec|\sum \omega_j C_j$ with non-trivial weights in contrast, there is no realistic chance to improve the performance guarantee of Theorem 3.3.8, unless P=NP. Thus for the problem of minimising the total weighted completion time it might be helpful to consider special cases of AND/OR-networks. In Section 2.5 we have investigated critical jobs in AND/OR-networks with in-tree structure and N-free AND/OR-networks. The in-tree structure changed the complexity status of critical jobs drastically. These restricted structures might also ease the approximability of the total weighted completion time.

3.3.2 Scheduling on Identical Parallel Machines

We now turn our attention to scheduling precedence constrained jobs on m identical parallel machines. We will restrict our focus on a short overview about the results of scheduling jobs on m machines with standard precedence constraints.

The problem of minimising the total weighted completion time of a set of jobs on two identical parallel machines with or without preemption was proved to be (weakly) NP-hard by Bruno, Coffman Jr., and Sethi (1974) and Lenstra, Rinnooy Kan, and Brucker (1977). Kawaguchi and Kyan (1986) showed that list scheduling with Smith's rule is a $\frac{\sqrt{2}+1}{2}$-approximation algorithm for $P||\sum \omega_j C_j$. Only recently, Skutella and Woeginger (2000) proposed a polynomial time approximation scheme (PTAS) for the basic problem $P||\sum \omega_j C_j$. In general, a strongly NP-hard optimisation problem cannot possess an FPTAS unless P=NP, see Garey and Johnson (1979). For the strongly NP-hard parallel machine scheduling problem, $P||\sum \omega_j C_j$, this means that the PTAS is best possible.

For the case that the jobs have non-trivial release dates, Hall, Schulz, Shmoys, and Wein (1997) provide an approximation algorithm that is basically a list scheduling guided by the solution of an LP-relaxation to the problem. This approach yields a $(4 - \frac{1}{m})$-approximation for the problem $P|r_j|\sum \omega_j C_j$, where m is again the number of machines that are available. This algorithm is slightly better and much simpler than the $(4+\varepsilon)$-approximation proposed by Hall, Shmoys, and Wein (1996) for $P|r_j|\sum \omega_j C_j$. Chakrabarti, Phillips, Schulz, Shmoys, Stein, and Wein (1996) applied a new conversion idea to the solution of the LP-relaxation proposed by Hall et al. (1997) achieving an improved performance guarantee of 3.5 for $P|r_j|\sum \omega_j C_j$. A better bound of 2 has been presented by Schulz and Skutella (2002). Nevertheless, all those algorithm for $P|r_j|\sum \omega_j C_j$ now get outperformed by the polynomial time approximation scheme proposed by Afrati et al. (1999), which is based on deliberate rounding and enumeration techniques.

If additional precedence constraints are involved, Hall et al. (1997) admit that they did not know how to prove a good performance guarantee by using simple list scheduling. Nevertheless, they propose a more sophisticated algorithm yielding a performance guarantee of 7 for the problem $P|prec, r_j|\sum \omega_j C_j$. This algorithm is again based on the optimal solution to an LP-relaxation to the problem. According to the optimal completion time of the LP-relaxation the jobs are divided into intervals of geometrically increasing length. The jobs belonging to one such interval are then scheduled with Graham's list scheduling. The *Interval-Schedule*, as Hall et al. name it, was the first known approximation algorithm for $P|prec, r_j|\sum \omega_j C_j$ achieving a constant performance guarantee. This algorithm has later been turned into a 5.33-approximation by Chakrabarti, Phillips, Schulz, Shmoys, Stein, and Wein (1996).

The solution to the LP-relaxation used by Hall et al. (1997) and Chakrabarti et al. (1996) has been turned into a simple 4-approximation for $P|prec, r_j|\sum \omega_j C_j$ by Munier, Queyranne, and Schulz (1998) by taking the midpoints of the LP-solution instead of the completion times to create the priority list for Graham's list scheduling. The algorithm even works in the presence of non-negative precedence delays d_{ij}. Precedence delays d_{ij} are a generalisation of standard precedence constraints $(i, j) \in E$ with the following meaning: after the completion of i, another d_{ij} time units have to elapse before job j can be started. Precedence delays are used to model sequence dependent setup or changeover times, for example. For the problem without release dates, that is $P|prec|\sum \omega_j C_j$, Munier et al. (1998) prove that scheduling by LP-midpoints gives a slightly tighter guarantee of $4 - \frac{2}{m}$.

Chekuri, Motwani, Natarajan, and Stein (2001) present several approximation algorithms for minimising the total weighted completion time of a set of jobs on identical parallel machines. These algorithms are based on the solution of a preemptive one-machine relaxation of the problem. For the problem $Pm|r_j|\sum \omega_j C_j$ a $(3 - \frac{1}{m})$-approximation is obtained by scheduling the jobs in non-decreasing order of their completion times in the one-machine relaxation. This simple list scheduling also works for precedence constrained jobs, if they have unit processing times. If the jobs have arbitrary processing times and are restricted by precedence constraints, a more sophisticated strategy is proposed to find a balance between leaving machines deliberately idle and possibly blocking them with long jobs. Given a ρ-approximation for the one-machine schedule, the conversion algorithm presented by Chekuri et al. gives an m-machine schedule that is within a factor $(1 + \beta)\rho + (1 + \frac{1}{\beta})$ of an optimum m-machine schedule, for any constant $\beta > 0$. The algorithm is also applicable in the presence of release dates, achieving the same bound for $Pm|prec, r_j|\sum \omega_j C_j$. In the case that it is possible to compute an optimal one-machine schedule, the conversion algorithm gives a 4-approximation for the m-machine schedule by setting $\beta = 1$. Thus the conversion algorithm of Chekuri et al. is a 4-approximation for minimising the total completion time on identical parallel machines when the precedence graph is a tree or a series-parallel graph and the jobs have release dates. Furthermore, the algorithm is a 2-approximation for $Pm|in\text{-}tree|\sum \omega_j C_j$. While simple list scheduling achieves only a guarantee of 3 for the problem with release dates but without precedence constraints, the conversion algorithm achieves a 2.83-approximation ratio for $Pm|r_j|\sum \omega_j C_j$. The algorithms and results of Chekuri et al. have previously been published in Chekuri, Motwani, Natarajan, and Stein (1997) and in Chekuri (1998).

There is little that can be said about the problem when AND/OR precedence constraints are imposed among the jobs. The n-approximation of list scheduling with shortest processing time rule for $1|ao\text{-}prec|\sum \omega_j C_j$ proved in Section 3.3.1 carries over to the case of parallel machines but we do not know how to im-

prove this bound. Neither of the approaches that led to approximation algorithms for standard precedence constraints seems to be useful for the general setting of AND/OR precedence constraints. There is a bunch of open questions to be answered for scheduling jobs in AND/OR-networks and we are very much interested in further success in this field of research.

CHAPTER 4

LOCAL SEARCH

A standard tool for handling hard combinatorial optimisation problems are local search heuristics. The key idea is to consider the space of all solutions of the problem and to define a neighbourhood relation on this space. A local search starts from one solution of the problem and inspects its neighbourhood for another solution. The goal is to find an optimal solution by iterating this procedure, where various rules may guide the choice of the neighbour. In the literature one can find a variety of algorithms based on this basic idea such as taboo search, simulated annealing, genetic algorithms and some variants of neural networks. For an intensive introduction into, further references to, and various applications of local search we refer to Aarts and Lenstra (1997). Neighbourhood search algorithms, as they are also called, have been widely and successfully applied to many practical problems. Although they are in general difficult to analyse or have poor worst case performance they often yield near optimal solutions for large problem instances in very short time. This has made local search a widely accepted approach to handle many complex real-world problems. We will first give an introduction into local search in combinatorial optimisation. Thereafter we will present a canonical neighbourhood for AND/OR-networks. In section 4.3 we will prove that the set of feasible AND/OR-networks is connected under the described neighbourhood relation. In the last section we will discuss special search strategies.

4.1 Introduction to Local Search

We will present the basic definitions and discuss the questions that arise when applying local search. First of all we have to clarify what we mean by an instance of a combinatorial optimisation problem and a neighbourhood. A lot of the terminology used here is taken from Aarts and Lenstra (1997) and we will omit explicit citation in most cases.

Definition 4.1.1. *An* instance *of a combinatorial optimisation problem is a pair* $(\mathscr{S}, \gamma)$*, where* $\mathscr{S}$ *is the* solution set *and* $\gamma : \mathscr{S} \to \mathbb{R}$ *is the* objective function*. The problem is to find a globally* optimal solution*, i. e.* $S^* \in \mathscr{S}$ *such that for all* $S \in \mathscr{S}$*,* $\gamma(S^*) \leq \gamma(S)$ *if the optimisation problem is a minimisation problem and*

$\gamma(S^*) \geq \gamma(S)$ *if it is a maximisation problem. Moreover,* $\gamma^\star = \gamma(S^*)$ *denotes the* optimal value, *and* $\mathscr{S}^* = \{S \in \mathscr{S} \mid \gamma(S) = \gamma^\star\}$ *denotes the* set of optimal solutions.

If not stated differently, we will assume that we are given a minimisation problem. In general, an instance $(\mathscr{S}, \gamma)$ of a combinatorial optimisation problem will not be given explicitly, i. e. by a list of all solutions, but in some compact way together with a polynomial time algorithm to evaluate a solution in this compact representation. The representation should enable fast determination of the neighbourhood of a solution. Actually, the neighbourhood is often described via specific features of the data representation.

Definition 4.1.2. *Let* $(\mathscr{S}, \gamma)$ *be a combinatorial optimisation problem. A* neighbourhood function *is a mapping* $\mathscr{N} : \mathscr{S} \to 2^{\mathscr{S}}$*, which defines for each solution* $S \in \mathscr{S}$ *a set* $\mathscr{N}(S) \subseteq \mathscr{S}$ *of solutions that are in some sense close to* S*. The set* $\mathscr{N}(S)$ *is the* neighbourhood *of solution* S*, each* $S' \in \mathscr{N}(S)$ *is a* neighbour *of* S *and we shall assume that* $S \in \mathscr{N}(S)$*, for all* $S \in \mathscr{S}$*.*

We will only consider symmetric neighbourhood functions, i. e. $S' \in \mathscr{N}(S)$ if and only if $S \in \mathscr{N}(S')$. The solution set $\mathscr{S}$ together with a neighbourhood function $\mathscr{N}$ can be represented by a *neighbourhood graph* $\mathscr{G} = (\mathscr{S}, E)$. The solutions are the vertices of the graph and there exists an edge $\{S, S'\} \in E$ if $S' \in \mathscr{N}(S)$. For symmetric neighbourhoods the neighbourhood graph is undirected. In the terminology of graph theory, local search is a traversal on the neighbourhood graph of the problem instance.

Iterative improvement is the basic version of local search. The algorithm starts with some initial solution S and searches the neighbourhood of S for a solution S' with better objective value $\gamma(S') < \gamma(S)$. If such a solution is found, the current solution is replaced by it, and the algorithm continues. If no better solution can be found, the algorithm stops. Obviously, iterative improvement will return a *local minimum* defined as follows.

Definition 4.1.3. *Let* $(\mathscr{S}, \gamma)$ *be a combinatorial minimisation [maximisation] problem and* $\mathscr{N}$ *be a neighbourhood function. A solution* $S \in \mathscr{S}$ *is a* local minimum [local maximum] *with respect to* $\mathscr{N}$*, if* $\gamma(S) \leq \gamma(S')$ $[\gamma(S) \geq \gamma(S')]$*, for all* $S' \in \mathscr{N}(S)$*.*

When designing a neighbourhood search, a crucial point is the choice of the neighbourhood function. The different properties of various neighbourhood functions influence strongly the performance of the local search algorithm. Among others we want to mention four of them here.

First of all, neighbourhoods differ in *ease*. When searching for a new neighbour it is a great benefit for the speed of the algorithm if the neighbourhood function allows a quick update of the objective function value instead of performing a complete recalculation.

Second, neighbourhoods differ in *size*. For a single solution the number of neighbours might be small or large. Small neighbourhoods allow for a quick search of the whole neighbourhood of a solution, while in a large neighbourhood one probably has to restrict to examine only parts of the neighbourhood of the given solution. On the other hand very small neighbourhoods tend to generate final solutions of poor quality.

Third, the quality of the solution produced depends on the underlying *topology* of the neighbourhood structure. The neighbourhood graph induces a distance measure: The number of neighbourhood moves needed to go from one solution to another, on the solution space. Assuming that we can embed the solution points in the plane such that the distances are correctly represented, we then can assign the objective function value of each solution as vertical component to the solution point. In this setting, the local search procedure moves around on this surface and the local optimum of the neighbourhood search corresponds to a local optimum in the classical continuous optimisation sense. Accordingly, if the surface is very bumpy, it might be difficult for the local search procedure to find a good solution, whereas if the surface is nice and smooth, the procedure can simply follow a descending path toward a very good solution.

Last but not least, the *connectivity* of the feasible solutions influences the quality of the generated solution. If the feasible solutions are disjoint, it might be necessary for the local search procedure to either accept infeasible solutions in between or to allow restarts from a different solution, to be able to find a good solution. On the other hand, if the space of feasible solutions is connected with respect to the neighbourhood function, the search procedure can be restricted to consider feasible solutions only, thus also influencing the size of the neighbourhood.

To overcome the problems originated by the chosen neighbourhood function, different search strategies have been developed. Even for the very simple iterative improvement procedure we can distinguish between *best* or *first* improvement. Best improvement searches the whole neighbourhood of a solution for the solution with the best objective function value, whereas first improvement checks the neighbours until it has found a neighbour with a better value than the current solution. Depending on the size and the topology of the neighbourhood, one or the other might be more efficient.

Concerning feasibility of solutions there are essentially four possible strategies to be considered. If the set of feasible solutions is connected one can simply *exclude* infeasible solutions. Otherwise, one could allow infeasible solutions but

penalise them by adding some penalty cost to the objective function. Another possibility is to *repair* a generated infeasible solution such that it becomes feasible or to leave it unaltered but assign an objective function value of its feasible *translated* version to it.

The most difficult problem arises from the topology of the neighbourhood structure. Most of the time, it is already difficult to define an easily computable measure of smoothness. By definition, the iterative improvement procedure will always run into a local optimum, which might be of very poor quality. A simple restart procedure can be applied to ease the problem. Nevertheless, some more elaborate strategies have been proposed in the literature like the previously mentioned simulated annealing, taboo search, and genetic algorithms. We refer to Anderson, Glass, and Potts (1997) for an overview over and references to applications of local search in scheduling.

Simulated annealing is a randomised local search algorithm which accepts, in addition to better-value solutions, also worse-value solutions. If a better-cost solution is detected in the neighbourhood it is always accepted. If the selected solution is of worse value, it is accepted with a probability that is gradually decreased during the execution of the algorithm. Under certain mild conditions, the randomised nature of simulated annealing allows for asymptotic convergence to optimal solutions, unfortunately typically requiring exponential time. In practice it often has been successfully applied as a heuristic with much faster convergence rates.

Taboo search selects as the new solution a legal neighbour of the current solution with best objective function value among all legal neighbours. This value might be worse than the value of the current solution. To determine a legal neighbour, the algorithm keeps a regularly updated list of currently illegal solutions. The *taboo list* prevents the procedure from running into cycles. To prevent the search from running into dead ends, it may also accept a solution from the taboo list, if it is good enough in some sense. Taboo search has been applied successfully for many problems, but there is no theoretical guideline for it and it has to be tailored to the specific problem details.

Genetic local search is based on the concepts of evolution theory. First, an *initial* set (population) of n solutions is generated. Then, local search is applied to find n local optima, the first *improving* step. Next, in the *recombining* step, a set of m offspring solutions is added to the present population. In another *improving* step, local search is used to replace the m offspring solutions by m local optima. Finally, the population is reduced to its original size by *selecting* n solutions from the current set. The process, without the initial step, is repeated until some stop

criterion is satisfied. The main potential of the genetic local search lies in the recombination step, where it can take advantage of the fact that several local optima are available.

In general there are no rules to specify the strategy for a successful local search algorithm. It heavily depends on the specific details of the combinatorial optimisation problem under consideration which procedure should be applied. There are numberless variants and combinations of the stated basic strategies that have been developed to exploit the properties of the specific problem successfully.

Neighbourhood search is a quite powerful tool to attack hard optimisation problems. In the following we will apply local search to resource constrained project scheduling.

4.2 A Canonical Neighbourhood for AND/OR-Networks

We consider an instance of the resource constrained project scheduling problem. It shall be given by a set V of jobs, a digraph $G = (V, E)$ depicting the precedence constraints, and a set $\mathcal{F} = \{F_1, \ldots, F_f\}$ of minimal forbidden sets representing the resource constraints. With each job $j \in V$ a strictly positive processing time p_j is associated and the goal is to find a schedule $S = (S_1, \ldots, S_n)$ minimising the makespan $C_{\max} = \max\{S_j + p_j \mid j \in V\}$. Minimising the makespan of a set of jobs restricted by minimal forbidden sets is a generalisation of m-machine scheduling and thus NP-hard in the strong sense, even without precedence constraints. For the reduction, for any fixed $m \geq 2$ set $\mathcal{F} = \{F \subseteq V \mid |F| = m + 1\}$. Schäffter (1997) proves that the problem remains NP-complete in the strong sense even for unit processing times and forbidden sets of size two, by showing that the k-colourability problem of graphs is a special case.

In Section 1.1.1 we have recorded that resource conflicts described by minimal forbidden sets can be resolved by selecting a waiting job for each of them. Such a selection can be displayed by an AND/OR-network. We have seen that deciding feasibility of a selection, respectively AND/OR-network, can be solved in linear time. Computing an earliest start schedule which minimises the makespan can be done in polynomial time. Thus, once we have made the selection of waiting jobs for the forbidden sets, there is nothing hard about optimisation any more. The hard part is to find the optimal selection for the given set $\mathcal{F}$, that is a selection that minimises the makespan of its associated schedule among all feasible selections. At this point, local search comes into play.

We will define a quite natural neighbourhood for AND/OR-networks with the property that the set of feasible AND/OR-networks is connected. This means that it is possible to reach a (global) minimum by neighbourhood moves only

considering feasible solutions. We shortly recall some of the relevant notations and results we will need in the following. In addition we will not distinguish between a selection and its corresponding earliest start schedule and use the term S alternatively for both of them.

We will assume that the jobs $V = \{1, \ldots, n\}$ are topologically sorted with respect to the precedence digraph G. A schedule S, the earliest start schedule, is uniquely determined by a selection of waiting jobs $(j_1, \ldots, j_f)$ for the forbidden sets $\mathcal{F} = \{F_1, \ldots, F_f\}$. The AND/OR-network resulting from such a selection S will be called $N(S)$. We have already stated in Theorem 1.2.2 that

$$\begin{aligned} S \text{ feasible} &\Leftrightarrow \nexists \text{ a generalised cycle in } N(S) \\ &\Leftrightarrow \exists \text{ a realisation } R \text{ of } N(S) \\ &\Leftrightarrow \exists \text{ a linear realisation } L = j_1, \ldots, j_n \text{ of } N(S) \end{aligned}$$

We will denote the set of all feasible selections by $\tilde{\mathscr{S}}$. For any feasible selection $S \in \tilde{\mathscr{S}}$ let $L_1, \ldots L_{\ell(S)}$ be its linear realisations. The lexicographically minimal linear realisation of all linear realisations for selection S will be called the *minimal realisation* $\underline{L}$.

We have everything at hand to define a neighbourhood on the solution space, i. e. the set $\mathscr{S}$ of all possible earliest start schedules.

Definition 4.2.1. *For a given instance of the scheduling problem with minimal forbidden sets $\{F_1, \ldots, F_f\}$, let S and S' be the schedules corresponding to two selections $(j_1, \ldots, j_f)$ and $(j'_1, \ldots, j'_f)$ of waiting jobs. Then S' is a neighbour of S if and only if there exists exactly one i, $1 \leq i \leq f$, such that $j_i \neq j'_i$ and $j_k = j'_k$ for all $k \neq i$. We will refer to this neighbourhood relation as the* canonical neighbourhood.

In other words, two solutions are neighbouring if they differ in exactly one waiting job.

4.3 Properties of the Solution Space

Given the above canonical neighbourhood we will now analyse some properties of the resulting solution space.

Size. The solution space $\mathscr{S}$ is in general, that is if the number of forbidden sets f grows with the number of nodes n, exponentially large as we have altogether $|F_1| \cdots |F_f|$ many possible selections. The neighbourhood $\mathscr{N}(S)$ of a single solution $S \in \mathscr{S}$, in contrast, is of size polynomial in the input, $|\mathscr{N}(S)| = \sum_{i=1}^{f} |F_i| - 1$.

Ease. There exists a dijkstra-like, polynomial time algorithm to compute the earliest start time schedule for a given AND/OR-network, we have described it in Section 1.2.2. Thus it is possible to compute the value of the objective function of a neighbour in polynomial time by simply recomputing the earliest start schedule for the neighbouring AND/OR-network. With some additional extra space it might be possible to save some time in the computation of the neighbouring schedule. Nevertheless, in the worst case there does not seem to be a straight forward method to decrease the asymptotic time complexity of computing the earliest start schedule of a neighbouring network.

Topology. There is little that can be said about the topology of the solution space under the canonical neighbourhood. There might be a descending path, but it is also possible that there are many isolated local minima, which are difficult to reach from one another. We will discuss this topic later again in more detail.

Connectivity. As mentioned above, the canonical neighbourhood has a nice property with respect to connectivity of feasible AND/OR-networks, which is stated in the next theorem.

Theorem 4.3.1. *The set $\tilde{\mathscr{S}}$ of feasible selections is connected with respect to the canonical neighbourhood.*

The proof will be structured as follows. First we introduce a default feasible selection. Then we show that for any feasible solution, we can always construct a series of neighbourhood moves on the set of feasible solutions which ends at the default selection. As neighbourhood moves are undirected this gives us the desired result. Feasibility of a selection will be proved by a linear realisation. We need the following definition for the proof.

Definition 4.3.2. *Let $S = (j_1, \ldots, j_f)$ be a feasible selection and L a linear realisation of the resulting AND/OR-network $N(S)$. We say that waiting job $j \in F$ is* released by *$x \in F$, if x is the job in $F \setminus \{j\}$ that comes first in the linear realisation L.*

A feasible selection has a linear realisation and in a linear realisation, for every forbidden set F and its waiting job j, there exists a job $x \in F \setminus \{j\}$, that comes before j in the realisation. The first job x among those that come before j in the list, is said to release j, where $x \neq j$ by construction. This definition is an important ingredient for the construction in the following proof of the theorem.

Proof of Theorem 4.3.1. As mentioned above we will assume that the jobs $V = \{1, \ldots, n\}$ are topologically sorted according to the precedence digraph G. Thus,

if we select, for every forbidden set, the job with the largest number, i. e. $S^0 = (j_1^0, \ldots, j_f^0)$ with $j_i^0 = \max\{j_k | j_k \in F_i\}$, for all $i = 1, \ldots, f$, then selection S^0 is feasible with the minimal linear realisation $\underline{L}^0 = 1, \ldots, n$. Given any feasible selection S, we will show that there is a series of selections

$$S = S^q, S^{q-1}, \ldots, S^1, S^0$$

ending at the default selection S^0, such that

- $S^{i-1} \in \mathscr{N}(S^i)$, i. e. S^{i-1} is a neighbour of S^i and
- $S^i \in \tilde{\mathscr{S}}$, i. e. S^i is feasible,

for all $i = 1, \ldots, q-1$. We will give a proof actually constructing the sequence of neighbourhood moves. Let $S = (j_1, \ldots, j_f)$ be a feasible solution. We set $S^q = S$ and carry out the following steps.

1. Let $\underline{L}^q = v_1, \ldots, v_n$ be the minimal realisation of S^q.
   If $v_i = i$ for all $i = 1, \ldots n$, continue with Step 6.
   Otherwise, consider $i \in \{1, \ldots, n\}$ minimal with the property that $v_i \neq i$. Obviously, i is the waiting job for some forbidden set $F \in \mathfrak{F}$ and $\underline{L}^q = v_1, \ldots, v_{i-1}, v_i, \ldots, i, \ldots, v_n$. By construction, the following holds:

   - $v_h = h$ for all $h = 1, \ldots, i-1$
   - $v_i > i = v_k$, with $i < k$

   From the choice of the linear realisation it follows that there exists some $F = F(i)$ and some j, $i \leq j < k$, such that $\{v_k, v_j\} \subseteq F$ and $v_k = i$ is released by v_j.

2. Consider the maximal j with the property that there exists a forbidden set $F \in \mathfrak{F}$ such that

   - $\{v_k, v_j\} \subseteq F$ and (*)
   - v_k is released by v_j. (**)

   Due to the minimality of i, the maximality of j, and the lexicographical minimality of the realisation $\underline{L}^q$, it follows that $j = k-1$ and thus

$$\underline{L}^q = \underset{\substack{\shortparallel \\ 1}}{v_1}, \ldots, \underset{\substack{\shortparallel \\ i-1}}{v_{i-1}}, \underbrace{v_i, \ldots, v_j}_{>i}, \underset{\substack{\shortparallel \\ i}}{v_k}, \underbrace{v_{k+1}, \ldots, v_n}_{>i}$$

   Let $F_1, \ldots F_r$, $r \geq 1$, be the forbidden sets with the two properties (*) and (**). We have to distinguish between the cases that $r > 1$ and $r = 1$.

3. If $r > 1$, by the inclusion-minimality of the minimal forbidden sets, it follows that $|F_\rho| \geq 3$ for all $\rho = 1, \ldots, r$. Thus, for all $\rho = 1, \ldots, r$, there exists $v^\rho \in F_\rho \setminus \{v_k, v_j\}$, $v^\rho > i$. Every set F_ρ has property (**), that is, v_k is released by v_j. Thus we can conclude that v^ρ appears behind v_j and v_k in the linear list $\underline{L}^q$ (as v_k would be released by v^ρ if it appeared in front of v_j in the list). Therefore, for every forbidden set F_ρ, $\rho = 1, \ldots, r$, we can exchange waiting job v_k against v^ρ. This yields a series of selections $S^q, S^{q-1}, \ldots, S^{q-r}$ such that $S^{q-\rho} \in \mathscr{N}(S^{q-\rho+1})$ for all $\rho = 1, \ldots, r$. By the choice of the exchanged waiting conditions, $\underline{L}^q$ is a linear realisation for all of them which immediately implies that they are all feasible. In addition,

$$L^{q-r} = v_1, \ldots, v_{i-1}, \ldots, v_k, v_j, v_{k+1}, \ldots, v_n$$

   is a linear realisation for S^{q-r}. As $v_k = i < v_j$, we know that $L^{q-r} < \underline{L}^q$.

4. If $r = 1$, that is, there is only one forbidden set with the properties (*) and (**), we replace waiting condition (F, v_k) with (F, v_j). It is easy to see that $L^{q-1} = L^{q-r}$ from Step 3 is a linear realisation of the new selection S^{q-1}, proving its feasibility.

5. Return to Step 1 with $S^q := S^{q-r}$.

6. By now, selection S^q has minimal realisation $\underline{L}^q = \underline{L}^0$. The only thing that is left to do, is to set the waiting job in each forbidden set to the maximum job of the set. This completes the construction.

It now remains to show that this whole procedure terminates with the default selection. First let us consider one iteration from Step 1 to Step 5. In any case we carry out r neighbourhood moves, where r is bounded from above by the number of forbidden sets in $\mathcal{F}$, $r \leq f$, which is of course a bounded number. Second, each iteration is started with selection S^q with minimal realisation $\underline{L}^q$ and ends with selection S^{q-r} with a linear realisation L^{q-r} which is strictly smaller than the minimal realisation $\underline{L}^q$ of S^q. This implies, that the procedure can only perform a bounded number of iterations, which completes the proof. □

We want to close this chapter with an overview over possible local search strategies.

4.4 Local Search and Longest Paths

A scheduling problem that has been approached by local search quite successfully is the job shop scheduling problem. We will introduce this problem and explain in which way it is related to scheduling with AND/OR-networks.

The job shop problem can be described as follows. Given is a set $\mathcal{M} = \{M_1, \ldots, M_m\}$ of machines and a set $\mathcal{J} = \{J_1, \ldots, J_n\}$ of n jobs. Each job J_j consist of a chain of n_j operations $O_{j1}, \ldots, O_{jn_j}$, where operation O_{ji} must be processed on machine $M_{ji} \in \mathcal{M}$ during p_{ji} time units without interruption. The set of all operations O_{ji}, $j = 1, \ldots, n$ and $i = 1 \ldots, n_j$, shall be denoted by $\mathcal{O}$. The operations O_{ji} of one job J_j have to be processed one after another in the given order $O_{j1}, \ldots, O_{jn_j}$. Except for the fixed assignment of the operations to the machines, the job shop problem can be seen as a standard precedence constrained scheduling problem on $\sum_{j=1}^{n} n_j$ jobs, where the precedence constraints consist of n parallel chains. This is a very restricted structure of the precedence constraints. More generalised models have been considered by De Bontridder (2000). In these generalised models, the strict chain structure of the operations is replaced by arbitrary (acyclic) precedence constraints, which correspond to standard precedence structures of project scheduling problems. The difference between the job shop problem and standard precedence constrained scheduling problems lies in the machine assignment which makes the job shop problem one of the hardest scheduling problems to solve. Another generalisation has been considered by Mastrolilli and Gambardella (2000). Here, the strict machine assignment is weakened to the condition that each operation O_{ji} has to be performed by one machine M_k out of a set $M_{ji} \subseteq \mathcal{M}$ for p_{jik} time units. We will omit any further details of those special cases. The objective of the job shop problem is to find a schedule that minimises the length of the schedule, that is the makespan.

Some special cases of the job shop problem can be solved in polynomial time; Hefetz and Adiri (1982) propose an optimal algorithm for the job shop problem on two machines and with unit processing times for all operations, for example. Nevertheless it is already NP-hard in the strong sense to decide whether there exists a schedule of length less or equal to 4, see Williamson, Hall, Hoogeveen, Hurkens, Lenstra, Sevast'janov, and Shmoys (1996). This implies that there is no approximation algorithm with a performance ratio strictly better that $5/4$ unless P=NP.

Roy and Sussmann (1964) proposed a useful representation of an instance of the job shop problem, the so-called *disjunctive graph* model. The disjunctive graph $G = (V, A, E)$ is a mixed graph, that is a graph containing directed and undirected edges, defined as follows:

- $V = \mathcal{O} \cup \{s, t\}$, where s and t are the dummy start and end vertices as usual.
- A represents the given precedence constraints between the operations, that is for all $j = 1, \ldots, n$ the directed edges (O_{ji}, O_{ji+1}), $i = 1, \ldots, n_j - 1$.
- $E = \{\{O_{ji}, O_{j'i'}\} \mid M_{ji} = M_{j'i'}\}$ represents the machine constraints. There

is an undirected edge between any two operations if they have to be scheduled on the same machine.

In the disjunctive graph, a solution to the job shop problem can be generated by orienting the edges in E and thereby fixing the order in which each machine processes the jobs that are assigned to it. For each set $E' \subseteq E$, an *orientation* on E' is a function $\Omega : E' \to \mathcal{O} \times \mathcal{O}$ such that $\Omega(\{O_{ji}, O_{j'i'}\}) \in \{(O_{ji}, O_{j'i'}), (O_{j'i'}, O_{ji})\}$ for each $\{O_{ji}, O_{j'i'}\} \in E'$. An orientation is a *complete orientation*, if $E' = E$, otherwise we call it a *partial orientation*. Orientation $\Omega(E')$ is feasible if the digraph $(V, A \cup \Omega(E'))$ is acyclic. For a feasible orientation, an earliest start schedule and thus the makespan can be computed by a standard longest path algorithm. Obviously, there is a one-to-one correspondence between feasible schedules for an instance and feasible complete orientations of the edges in E. Thus the minimisation problem can also be formulated in terms of the disjunctive graph model $G = (V, A, E)$: Find a complete orientation Ω of E that minimises the longest path in $(V, A \cup \Omega(E))$.

A disjunctive graph can also be interpreted as a resource constrained project scheduling instance where the resource constraints are given by a set E of two-element minimal forbidden sets. In this representation, the orientation of an edge $\{u, v\} \in E$ in one direction corresponds to selecting one of the two jobs as the waiting job. Thus an oriented instance $(V, A \cup \Omega(E))$ of the job shop scheduling problem corresponds to an AND/OR-network $(V \cup W, A)$ where for every directed edge (u, v) in $\Omega(E)$ there exists a waiting condition $(\{u\}, v) \in W$.

In the disjunctive graph formulation, local search becomes a natural approach to solve the problem. Consider the set of all orientations, or the set of all feasible orientations. The most simple neighbourhood function defines a neighbour of one orientation by re-orienting one of the edges in E. Of course, there is a variety of other possibilities for defining a neighbourhood function. Vaessens, Aarts, and Lenstra (1996) provide a thorough overview over local search in job shop scheduling. The paper gives an introduction into the job shop problem, the different neighbourhood functions that have been introduced, as well as an overview over the various local search algorithms that have been proposed. They conclude their survey with a comparison of the computational results of the presented local search algorithms. The neighbourhood search algorithms presented cover nearly all possible variations of this approach, from shifting bottleneck functions, threshold algorithms including simulated annealing, taboo search, to genetic algorithms, as well as several hybrid approaches. We refer to the comprehensive paper of Vaessens, Aarts, and Lenstra (1996) for further details.

Most of the proposed algorithms have been guided by the following properties of the job shop scheduling problem observed by Balas (1971).

Lemma 4.4.1. *For a given instance of the job shop problem represented by the disjunctive graph (V, A, E), the following holds:*

a) Given a feasible orientation, reversing an oriented edge on a longest path in the corresponding digraph results again in a feasible orientation.

b) If reversing an oriented edge of a feasible orientation Ω that is not on a longest path results in a feasible orientation Ω', then the makespan of Ω' is at least as long as the makespan of Ω.

In Vaessens, Aarts, and Lenstra (1996) another two items are listed which describe two more criteria for reversing edges such that the makespan cannot decrease. These will not be of interest in the following.

Note that reversing an oriented edge corresponds to exchanging the waiting job in the AND/OR-network representation. As all minimal forbidden sets are two-element sets, an OR-node $w = (\{u\}, v)$ can be identified with the edge (u, v) and the AND/OR-network corresponds to a standard acyclic digraph. Therefore a longest path in such an AND/OR-network is the same as a longest path in a digraph and an OR-node is on a longest path, if the edge (u, v) is on a longest path. We can rewrite Property a of Lemma 4.4.1 for AND/OR-networks as follows: Given a feasible AND/OR-network, exchanging the waiting job of an OR-node on a longest path, respectively the waiting condition it represents, results in a feasible AND/OR-network. Property b can be rephrased for AND/OR-networks accordingly.

Let us come back to scheduling with general AND/OR-networks. Inspired by the successful application of local search to the job shop problem we would like to find similar properties of neighbouring AND/OR-networks that could guide a promising strategy. Therefore we would like to identify longest paths in AND/OR-networks, which brings us back to the notion of criticality. By Definition 2.2.2, a path-critical set is an inclusion-minimal set of jobs such that a small decrease of the processing times of the jobs in its complement does not affect the makespan. This property seems to be similar to Property b of Lemma 4.4.1 for the job shop problem. Nevertheless, in general AND/OR-networks this is not the case. Consider the two AND/OR-networks N and N' of Figure 4.1. The numbers next to the nodes represent the processing times of the jobs, $p = (2, 2, 1, 2, 3, 1)$. In the first network, N, the makespan is $C_{\max} = 5$ and there exists one path-critical set $U^P = \{4, 5\}$. The closure of U^P contains the OR-node w_2. The other OR-node, w_1, is not enclosed in any critical set. AND/OR-network $N' \in \mathscr{N}(N)$ can be obtained by exchanging waiting job 3 for the forbidden set F_1 against job 2. The makespan of N' is $C_{\max} = 4$, which is smaller than the makespan of N. This shows that there is no direct correspondence to Property b of the job shop problem. Note that the network N in Figure 4.1 is constructed such that turning

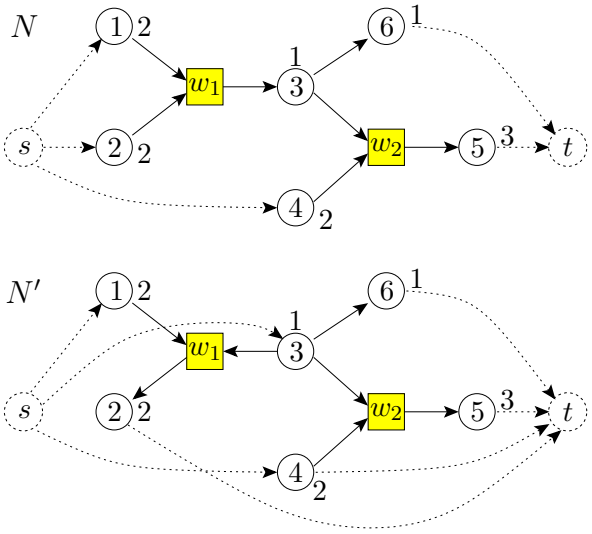


Figure 4.1: Two neighbouring AND/OR-networks N and N' with processing time vector $p = (2, 2, 1, 2, 3, 1)$. In N the makespan is $C_{\max}(N) = 5$ and there is one path-critical set $U^P = \{4, 5\}$. In network N' the makespan is $C_{\max}(N') = 4$.

around the edges on the critical path, that is exchanging waiting job 5 against 4 for the forbidden set F_2, does not improve the makespan. As U^P is also the only bulk-critical set of N, the same argumentation holds for bulk-critical sets instead of path-critical sets.

Now consider Property a in Lemma 4.4.1. Again it is easy to find an example which illustrates that there is no direct correspondence for AND/OR-networks. The following example will illustrate that turning around edges on a longest path, respectively a path- or bulk-critical set does not necessarily result in a feasible network.

In the scheduling instance depicted in Figure 4.2, a sequence of three neighbouring AND/OR-networks with processing time vector $p = (1, 1, 1, 1, 2)$ is given. In Network N and $N' \in \mathscr{N}(N)$, the makespan is 4, which is not optimal. In network N, there is exactly one "longest path", that is $\{3, 4, 5\}$, which is a path- as well as a bulk-critical set. The OR-nodes w_3 and w_4 are both enclosed in the closure of the path- and the bulk-critical set. Consider the minimal forbidden set F_3 corresponding to w_3. If we turn around the two tight edges incident to w_3 and thereby exchange waiting job 4 against job 3, this results in an infeasible AND/OR-network. The nodes in $(V \setminus \{5\}) \cup W$ form a generalised cycle in this case. The same holds for the forbidden set F_4 corresponding to w_4. Nevertheless, we can exchange the waiting job 4 of F_3 for example against job 1. This gives us network N' with the same makespan. In the neighbourhood of N', there ex-

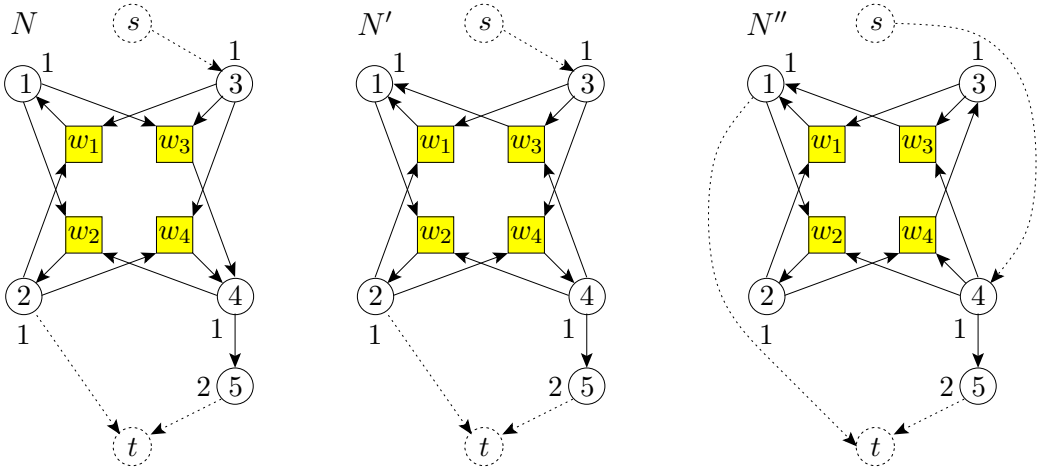


Figure 4.2: Three neighbouring AND/OR-networks N, $N' \in \mathscr{N}(N)$ and $N'' \in \mathscr{N}(N')$ with processing time vector $p = (1, 1, 1, 1, 2)$. The makespan of network N and N' is $C_{\max} = 4$, where the last network N'' has an improved makespan of 3.

ists a network, namely N'' with a better makespan and we reach it by exchanging waiting job 4 of F_4 against job 3.

The presented examples show that there is no direct correspondence between the properties of longest paths in the job shop scheduling problem and path-critical or bulk-critical sets in AND/OR-networks. Thus, criticality does not seem to help for guiding a local search strategy, at least not in this straightforward manner. We still think that neighbourhood search is a promising approach to find optimal or near optimal solutions for the optimisation problem in AND/OR-networks, but there is no obvious guidance for a specific strategy. Maybe more sophisticated neighbourhood functions would help. In any case there is a number of open questions to be answered and a variety of possible approaches to be pursued.

BIBLIOGRAPHY

Aarts, E. and J. K. Lenstra (Eds.) (1997). *Local Search in Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons.

Adelson-Velsky, G. M. and E. Levner (1999, November). Project scheduling in AND/OR graphs: A generalization of Dijkstra's algorithm. Technical report, Department of Computer Science, Holon Academic Institute of Technology, Holon, Israel.

Afrati, F., E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko (1999). Approximation schemes for minimizing average weighted completion time with release dates. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS'99)*, 32–43.

Anderson, E. J., C. A. Glass, and C. N. Potts (1997). Machine scheduling. In E. Aarts and J. K. Lenstra (Eds.), *Local Search in Combinatorial Optimization*, Wiley-Interscience Series in Discrete Mathematics and Optimization, Chapter 11, 361–414. John Wiley & Sons.

Arora, S., L. Babai, J. Stern, and Z. Sweedyk (1993). The hardness of approximate optima in lattices, codes and linear equations. In *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, 724–733.

Arora, S., L. Babai, J. Stern, and Z. Sweedyk (1997). The hardness of approximate optima in lattices, codes and systems of linear equations. *Journal of Computer and System Science 54*, 317–331. A preliminary version has been publishes as Arora, Babai, Stern, and Sweedyk (1993).

Arora, S. and C. Lund (1997). Hardness of approximation. In D. S. Hochbaum (Ed.), *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company.

Ausiello, G., P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi (1999). *Complexity and Approximation*. Springer.

Ausiello, G., A. d'Atri, and M. Protasi (1980). Structure preserving reductions among convex optimization problems. *Journal of Computer and System Science 21*, 136–153.

Ausiello, G., R. Giaccio, G. F. Italiano, and U. Nanni (1992). Optimal traversal of directed hypergraphs. Technical Report TR-92-073, Berkeley, CA.

Ausiello, G., G. F. Italiano, and U. Nanni (1998). Hypergraph traversal revisited: Cost measures and dynamic algorithms. In L. Brim, J. Gruska, and J. Zlatuska (Eds.), *Mathematical Foundations of Computer Science 1998*, Volume 1450 of *LNCS*, 1–16. Springer. Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS'98).

Balas, E. (1971). Project scheduling with resource constraints. In E. M. L. Beale (Ed.), *Applications of Mathematical Programming*, 187–200. The English University Press, London.

Bollobás, B. (1990). *Graph theory : an introductory course*, Volume 63 of *Graduate texts in mathematics*. Springer.

Brucker, P. (1998). *Scheduling Algorithms*. Springer Verlag.

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

Cambini, R., G. Gallo, and M. G. Scutellá (1997). Flows on hypergraphs. *Mathematical Programming 78*, 195–217.

Chakrabarti, S., C. A. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, and J. Wein (1996). Improved scheduling algorithms for minsum criteria. In F. M. auf der Heide and B. Monien (Eds.), *Automata, Languages and Programming*, Volume 1099 of *LNCS*, 646–657. Springer. Proceedings of the 23rd International Colloquium on Automata, Languages and Programming (ICALP'96).

Chekuri, C. (1998, August). *Approximation Algorithms for Scheduling Problems*. Dissertation, Department of Computer Science, Stanford University.

Chekuri, C., R. Johnson, R. Motwani, B. Natarajan, B. R. Rau, and M. Schlansker (1996). Profile-driven instruction level parallel scheduling with application to super blocks. In *Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-29)*, 58–67.

Chekuri, C. and R. Motwani (1999). Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *Discrete Applied Mathematics 98*, 29–38.

Chekuri, C., R. Motwani, B. Natarajan, and C. Stein (1997). Approximation techniques for average completion time scheduling. In *Proceedings of the*

8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'97), 609–618.

Chekuri, C., R. Motwani, B. Natarajan, and C. Stein (2001). Approximation techniques for average completion time scheduling. *SIAM Journal on Computing 31*, 146–166. A preliminary version was published in Chekuri, Motwani, Natarajan, and Stein (1997).

Chudak, F. and D. S. Hochbaum (1999). A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine. *Operations Research Letters 25*, 199–204.

De Bontridder, K. M. J. (2000). Minimizing total weighted tardiness in a generalized job shop. Technical report, Department of Mathematics and Computing Science, Technische Universiteit Eindhoven.

Diestel, R. (2000). *Graph theory*, Volume 173 of *Graduate texts in mathematics*. Springer.

Dinur, I. and S. Safra (1999). On the hardness of approximating label-cover. Electronic Colloquium on Computational Complexity (ECCC) Technical Report TR99-015, School of Mathematical Sciences, Tel Aviv University.

Fujii, M., T. Kasami, and K. Ninomiya (1969). Optimal sequencing of two equivalent processors. *SIAM Journal on Applied Mathematics 17*, 784–789. Erratum. SIAM Journal on Applied Mathematics 20 (1971) 141.

Fulkerson, D. R. (1961). A network flow computation for project cost curves. *Management Science 7*, 167–178.

Fulkerson, D. R. (1971). Blocking and anti-blocking pairs of polyhedra. *Mathematical Programming 1*, 168–194.

Gallo, G., C. Gentile, D. Pretolani, and G. Rago (1998). Max Horn SAT and the minimum cut problem on directed hypergraphs. *Mathematical Programming 80*, 213–237.

Gallo, G., G. Longo, S. Pallottino, and S. Nguyen (1993). Directed hypergraphs and applications. *Discrete Applied Mathematics 42*, 177–201.

Gallo, G. and M. G. Scutellá (1999). Directed hypergraphs as a modelling paradigm. Technical Report TR-99-02, Dipartimento di Informatica, Università di Pisa.

Garey, M. J. and D. S. Johnson (1979). *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. New York: Freemann.

Gillies, D. W. and J. W.-S. Liu (1991). Greed in recource scheduling. *Acta Informatica 28*, 755–775.

Gillies, D. W. and J. W.-S. Liu (1995). Scheduling tasks with AND/OR precedence constraints. *SIAM Journal on Computing 24*, 797–810.

Goemans, M. X., M. Queyranne, A. S. Schulz, M. Skutella, and Y. Wang (2002). Single machine scheduling with release dates. *SIAM Journal on Discrete Mathematics 15*, 165–192.

Goldwasser, M. H. and R. Motwani (1997). Intractability of assembly sequencing: Unit disks in the plane. In *Algorithms and Data Structures*, Volume 1272 of *LNCS*, 307–320. Springer. Proceedings of the 5th annual Workshop on Algorithms and Data Structures (WADS'97).

Goldwasser, M. H. and R. Motwani (1999). Complexity measures for assembly sequences. *International Journal of Computational Geometry & Applications 9*, 371–417.

Goossens, M., F. Mittelbach, and A. Samarin (1994). *The LaTeX Companion*. Addison-Wesley.

Graham, R. L. (1966). Bounds for certain multiprocessing anomalies. *Bell System Technical Journal 45*, 1563–1581.

Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics 17*, 416–429.

Graham, R. L., E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics 5*, 287–326.

Hall, L. A., A. S. Schulz, D. B. Shmoys, and J. Wein (1997). Scheduling to minimize average completion time: off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 513–549.

Hall, L. A., D. B. Shmoys, and J. Wein (1996). Scheduling to minimize weighted completion time: off-line and on-line algorithms. extended abstract. In *Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms (SODA'96)*, 142–151.

Hefetz, N. and I. Adiri (1982). An efficient optimal algorithm for the two-machines unit-time jobshop schedule-length problem. *Mathematics of Operations Research 7*, 354–360.

Hennessy, J. L. and T. Gross (1983). Postpass code optimization of pipeline constraints. *ACM Transactions on Programming Languages and Systems (TOPLAS) 5*, 4220–448.

Hochbaum, D. S. (Ed.) (1997). *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company.

Horn, W. A. (1972). Single-machine job sequencing with treelike precedence ordering and linear delay penalties. *SIAM Journal on Applied Mathematics 23*, 189–202.

Hu, T. C. (1961). Parallel sequencing and assembly line problems. *Operations Research 9*, 841–848.

Jungnickel, D. (1991). *Graphen, Netzwerke und Algorithmen*. BI Wissenschaftsverlag Mannheim Leipzig Wien Zürich.

Kawaguchi, T. and S. Kyan (1986). Worst case bound of an lrf schedule for the mean weighted flow-time problem. *SIAM Journal on Computing 15*, 1119–1129.

Kelley, J. E. (1961). Critical path planning and scheduling: Mathematical basis. *Operations Research 9*, 296 – 320.

Khanna, S., M. Sudan, and L. Trevisan (1997). Constraint satisfaction: The approximability of minimization problems. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC'97)*, 11–20.

Knuth, D. E. (1977). A generalization of dijkstra's algorithm. *Information Processing Letters 6*, 1–5.

Kolliopoulos, S. G. and G. Steiner (2002). Partially-ordered knapsack and applications to scheduling. In R. H. Möhring and R. Raman (Eds.), *Algorithms - ESA 2002*, Volume 2461 of *LNCS*, 612–624. Springer. Proceedings of the 10th Annual European Symposium on Algorithms (ESA'02).

Korte, B. and J. Vygen (2000). *Combinatorial Optimization, Theory and Algorithms*, Volume 21 of *Algorithms and combinatorics*. Springer.

Lawler, E. L. (1978). Sequencing jobs to minimize total weighted completion time. *Annals of Discrete Mathematics 2*, 75–90.

Lawler, E. L., J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys (1993). Sequencing and scheduling: Algorithms and complexity. In *Logistics of Production and Inventory*, Volume 4 of *Handbooks in Operations Research and Management Science*, 445–522. North-Holland, Amsterdam.

Lehman, A. (1979). On the width-length inequality. *Mathematical Programming 17*, 403–417.

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

Lenstra, J. K., A. H. G. Rinnooy Kan, and P. Brucker (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics 1*, 343–362.

Margot, F., M. Queyranne, and Y. Wang (2000). Decompositions, network flows, and a precedence constrained single machine scheduling problem. Technical Report 2000-29, Department of Mathematics, University of Kentucky, Lexington.

Mastrolilli, M. and L. M. Gambardella (2000). Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling 3*, 3–20.

Möhring, R. H. (1989). Computationally tractable classes of ordered sets. In I. Rival (Ed.), *Algorithms and Order*, 105–193. Kluwer Academic Publisher.

Möhring, R. H. and F. J. Radermacher (1989). The order-theoretic approach to scheduling: The deterministic case. In R. Słowiński and J. Węglarz (Eds.), *Advances in Project Scheduling*, 29–66. Elsevier Science Publisher B. V.

Möhring, R. H., M. Skutella, and F. Stork (2000a). Forcing relations for AND/OR precedence constraints. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00)*, 235–236.

Möhring, R. H., M. Skutella, and F. Stork (2000b). Scheduling with AND/OR precedence constraints. Technical Report 689/2000, Technische Universität Berlin, Department of Mathematics, Germany. To appear in SIAM Journal on Computing.

Munier, A., M. Queyranne, and A. S. Schulz (1998). Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. In R. Bixby, E. Boyd, and R. Ríos-Mercado (Eds.), *Integer Programming and Combinatorial Optimization*, Volume 1412 of *LNCS*, 367–383. Springer. Proceedings of the 6th International Conference on Integer Programming and Combinatorial Optimization (IPCO).

Nemhauser, G. L. and L. A. Wolsey (1988). *Integer and Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons.

Phillips, C., C. Stein, and J. Wein (1995). Scheduling jobs that arrive over time. In J. R. Sack and N. Santoro (Eds.), *Algorithms and Data Structures*, Volume 955 of *LNCS*, 86–97. Springer. Proceedings of the 4th International Workshop on Algorithms and Data Structures (WADS'95).

Phillips, C., C. Stein, and J. Wein (1998). Minimizing average completion time in the presence of release dates. *Mathematical Programming 82*, 199–224. A preliminary version has been published as Phillips, Stein, and Wein (1995).

Prestel, A. (1992). *Einführung in die Mathematische Logik und Modeltheorie*, Volume 60 of *Aufbaukurs Mathematik*. Vieweg.

Queyranne, M. and A. S. Schulz (1994). Polyhedral approaches to machine scheduling. Technical Report 408, Departmen of Mathematics, Technical University of Berlin, Berlin, Germany.

Roy, B. and B. Sussmann (1964). Les problèmes d'ordonnancement avec constraints disjonctives. Note ds no. 9 bis, SEMA, Montrouge.

Schäffter, M. (1997). Scheduling with respect to forbidden sets. *Discrete Applied Mathematics 72*, 155–166.

Schrijver, A. (2003). *Combinatorial Optimization, Polyhedra and Efficiency*, Volume 24 of *Algorithms and Combinatorics*. Springer.

Schulz, A. S. (1996). Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds. In W. H. Cunningham, S. McCormick, and M. Queyranne (Eds.), *Integer Programming and Combinatorial Optimization*, Volume 1084 of *LNCS*, 301–315. Springer. Proceedings of the 5th International Conference on Integer Programming and Combinatorial Optimization (IPCO'96).

Schulz, A. S. and M. Skutella (1997). Random-based scheduling: New approximations and LP lower bounds. In J. Rolim (Ed.), *Randomization and Approximation Techniques in Computer Science*, Volume 1269 of *LNCS*, 119–133. Springer. Proceedings of the 1st International Symposium on Randomization and Approximation Techniques in Computer Science (Random'97).

Schulz, A. S. and M. Skutella (2002). Scheduling unrelated machines by randomized rounding. *SIAM Journal on Discrete Mathematics*.

Skutella, M. (2002). List scheduling in order of alpha-points on a single machine. Technical Report 734-2002, Technische Universität Berlin, Department of Mathematics. To appear in a book on Approximation and On-line Algorithms edited by members of the EU-project APPOL.

Skutella, M. and G. J. Woeginger (2000). A PTAS for minimizing the total weighted completion time on identical parallel machines. *Mathematics of Operations Research 25*, 63–75.

Smith, W. E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly 3*, 59–66.

Stork, F. (2001). *Stochastic Resource-Constrained Project Scheduling*. Ph. D. thesis, Technische Universität Berlin, Germany.

Stork, F. and M. Uetz (2000). On the representation of resource constraints in project scheduling. Technical Report 693/2000, Technische Universität Berlin, Department of Mathematics.

Trotter, W. T. (1992). *Combinatorics and Partially Ordered Sets*. The Johns Hopkins University Press.

Uetz, M. (2001). *Algorithms for Deterministic and Stochastic Scheduling*. Ph. D. thesis, Technische Universität Berlin, Germany.

Ullman, J. D. (1975). NP-complete scheduling problems. *Journal of Computer and System Science 10*, 384–393.

Vaessens, R. J., E. Aarts, and J. K. Lenstra (1996). Job shop scheduling by local search. *INFORMS Journal on Computing 8*, 302–317.

Vazirani, V. V. (2001). *Approximation Algorithms*. Springer.

Wegener, I. (1987). *The Complexity of Boolean Functions*. Wiley-Teubner Series in Computer Science. John Wiley & Sons.

Weiss, S. and J. E. Smith (1987). A study of scalar compilation techniques for pipelined supercomputers. In *Proceedings of the 2nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-II)*, 105–109.

West, D. B. (2001). *Introduction to graph theory*. Prentice-Hall.

Williamson, D. P., L. A. Hall, J. A. Hoogeveen, C. A. J. Hurkens, J. K. Lenstra, S. V. Sevast'janov, and D. B. Shmoys (1996). Short shop schedules. *Operations Research*.

Woeginger, G. J. (2001). On the approximability of average completion time scheduling under precedence constraints. In F. Orejas, P. Spirakis, and J. van Leeuwen (Eds.), *Automata, Languages and Programming*, Volume 2076 of *LNCS*, 887–897. Springer. Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP'01).

SYMBOL INDEX

$ao\text{-}prec$	AND/OR precedence constraints , page 16
$\mathcal{B}$	set of bulk-critical sets, page 33
$BL(V, \mathcal{X})$	blocking clutter of $(V, \mathcal{X})$, page 33
$B(T)$	time-cost trade-off curve, page 81
$\mathcal{C}$	set of cut-critical sets, page 33
C	vector of completion times, $C = (C_1, C_2, \ldots, C_n)$, page 13
C_j	completion time of job j, page 13
C_{max}	makespan, page 14
$\sum C_j$	total completion time, page 14
$\sum \omega_j C_j$	total weighted completion time, page 14
C_{st}	s-t-cut in a hypergraph, $C_{st} = (V_s, V_t)$, page 24
Σ_{st}	cutset of an s-t-cut C_{st}, page 24
$\mathcal{D}$	set of delay-critical sets, page 33
E	set of edges, page 11
E^{tight}	set of tight edges, page 20
ES	earliest start schedule, page 20
$\mathcal{F}$	set of minimal forbidden sets, page 11
F	minimal forbidden set $F \in \mathcal{F}$, page 11
$\mathscr{G}$	neighbourhood graph, $\mathscr{G} = (\mathscr{S}, E)$, page 114
G	precedence digraph $G = (V(G), E(G))$, page 10
H	directed hypergraph $H = (V, A)$, page 23

$H(N)$	hypergraph of an AND/OR-network N, $H(N) = (V, A)$, page 23
$ImPred(v)$	set of immediate predecessors of v, page 12
$ImPred^{tight}(v)$	set of tight immediate predecessors of v, page 20
$ImSucc(v)$	set of immediate successors of v, page 12
$ImSucc^{tight}(v)$	set of tight immediate successors of v, page 20
i, j	jobs, $i, j \in V$ (AND-nodes), page 10
κ_j	cost function of job j, $\kappa_j : [p_j^{min}, p_j^{max}] \rightarrow \mathbb{R}_{\geq}$, page 81
L	linear realisation of an AND/OR-network, linear list, page 18
$\underline{L}$	lexicographically minimal linear realisation of S, page 118
$\underline{L}^0$	lexicographically minimal linear realisation of S^0, page 120
$l_{uv}^{\max}$	maximum length of a u-v-path in a digraph, page 90
$\mathscr{N}$	neighbourhood function, $\mathscr{N} : \mathscr{S} \rightarrow 2^{\mathscr{S}}$, page 114
$\mathscr{N}(S)$	neighbourhood of solution S, page 114
N	AND/OR-network $N = (V \cup W, E)$, page 11
N^{tight}	tight sub-network (induced by E^{tight}), page 20
$N(E')$	edge induced sub-network of N, page 17
$N(V' \cup W')$	vertex induced sub-network of N, page 17
$\Omega(E)$	orientation of an edge set E, page 123
$\mathcal{P}$	set of path-critical sets, page 33
P	identical parallel machines, page 16
Pm	m identical parallel machines, page 16
P_{uv}	u-v-path, page 12
Π_{Rv}	hyperpath from a root set R to a node v, page 24
$Pred(v)$	set of all predecessors of v, $v \in Pred(v)$, page 12
$Pred^{tight}(v)$	set of tight predecessor of v, page 20

p_j	processing time of job j, page 10
$p_{\min}$	minimum processing time of a job in V, page 10
$p^{[U+\varepsilon]}$	increased processing time vector, page 29
$prec$	standard precedence constraints, page 16
R	realisation of an AND/OR-network, page 18
$\mathbb{R}_{\geq}$	set of non-negative real numbers, page 81
r_j	release date of job j, page 10
ρ	approximation guarantee of an algorithm, page 88
$\varrho(N)$	Max-Flow-Min-Cut gap of network N, page 37
$\varrho(n)$	Max-Flow-Min-Cut gap, page 37
$\mathscr{S}$	set of solutions, page 113
$\tilde{\mathscr{S}}$	set of feasible solutions, $\tilde{\mathscr{S}} \subseteq \mathscr{S}$, page 118
S	schedule (vector of starting times), $S = (S_1, \ldots, S_n)$, page 13
S^*	optimal schedule, $S^* = (S_1^*, \ldots, S_n^*)$, page 88
$S^{[U+\varepsilon]}$	schedule of $(N, p^{[U+\varepsilon]})$, page 29
S^0	selection where the maximal jobs are selected, page 120
$Succ(v)$	set of all successors of v, $v \in Succ(v)$, page 12
$Succ^{tight}(v)$	set of tight successor of v, page 20
s	start vertex $s = 0$, source of N, page 12
$size(N)$	size of AND/OR-network N, page 12
$slack(N, p, e)$	slack of an edge $e \in E$, page 21
$slack(N, p)$	slack of AND/OR-network (N, p), page 22
t	end vertex $t = n + 1$, the sink of N, page 12
$U^{\wedge}$	AND-closed set of (N, p), page 52
$U^{\vee}$	OR-closed set of (N, p), page 51

U^B	bulk-critical set, $U^B \subseteq V$, page 29
$\overline{U^B}$	closure of bulk-critical set U^B, page 43
U^C	cut-critical set, $U^C \subseteq V$, page 29
U^D	delay-critical set, $U^D \subseteq V$, page 29
U^P	path-critical set, $U^P \subseteq V$, page 29
$\overline{U^P}$	closure of path-critical set U^P, page 42
V	set of jobs, $V = \{1, \ldots, n\}$ (AND-nodes), page 10
$(V, \mathcal{X})$	clutter, $\mathcal{X} \subseteq 2^V$, page 33
v, u	vertices (jobs or waiting conditions), $v, u \in V \cup W$, page 12
W	set of waiting conditions (OR-nodes), page 10
w	waiting condition, $w = (X, j) \in W$ (OR-node), page 10
ω_j	weight of job j, page 10
$\xi^j(S)$	threshold of job j with respect to schedule S, page 98

INDEX

AND-closed, 52
AND-constraints, *see* constraints
AND-node, 10
AND/OR-network, 11
    acyclic, 12
    feasible, 18
    hypergraph of, 23
    size of, 12
    sub-network, 17
    tight sub-network, 20
AND/OR/skipped model, 93
AND/OR/unskipped model, 93
approximation algorithm, 88
available, 89

blocking clutter, *see* clutter

closure
    of a bulk-critical set, *see* critical
    of a path-critical set, *see* critical
clutter, 33
    blocking, 33
completion time, 13
constraints
    AND-constraints, 13
    machine constraints, 13
    OR-constraints, 13
    precedence, 10
    release constraints, 13
cost function, 81
critical
    bulk-critical, 29
      closed set, 43
    cut-critical, 29
    delay-critical, 29
    path-critical, 29
      closed set, 42
cut in a hypergraph, 24
cutset, 24
cycle
    in a hypergraph, 24
    in an AND/OR-network, 12

DEADLINE PROBLEM, 84
disjunctive graph, 122

earliest start schedule, 21

feasible
    AND/OR-network, 18
    schedule, 13
forbidden set, 11
    minimal, 11
fully polynomial time approximation
        scheme (FPTAS), 88

generalised cycle, 17
genetic local search, 116
Graham anomalies, 90
Graham's list scheduling, *see* list schedul-
        ing

head of a directed edge, 23
HITTING SET, 84
HORN-SAT, 65
hyperedge, 23
hypergraph, 23
    acyclic, 24
    of an AND/OR-network, 23
    sub-hypergraph, 24
hyperpath, 24

in-tree, 71
induced sub-network, 17
instance, 113

internal-tree, 103
iterative improvement, 114

LABEL COVER, 103
labelling, 103
leaf, 103
linear time-cost trade-off problem, 81
list scheduling, 89
local optimum, *see* solution
longest processing time (LPT) rule, 91

machine constraints, *see* constraints
makespan, 14
Max-Flow-Min-Cut gap, 37
Max-Flow-Min-Cut property, 33
Minimum Path Heuristic (MPH), 93
monotone function, 65
    Boolean , 65

N, 71
N-free, 71
neighbour, 114
neighbourhood, 114
    canonical, 118
    function, 114
    graph, 114

objective function, 13
OR-closed, 51
OR-constraints, *see* constraints
OR-node, 11
orientation, 123

path
    cardinality of, 12
    in a hypergraph, 24
      acyclic, 24
    in an AND/OR-network, 12
      acyclic, 12
    length of, 12, 90
performance guarantee/ratio, 88
polynomial time approximation scheme (PTAS), 88
precedence constraints, *see* constraints
precedence digraph, 10
predecessor, 12
    direct, 12
    tight, 20

realisation, 18
    linear, 18
    minimal, 118
redundant, 70
release constraints, *see* constraints
released by, 119

SAT, 60
    HORN type, 65
schedule, 13
    earliest start, 21
    feasible, 13
    optimal, 88
    partial, 89
SET COVER, 38
shortest processing time (SPT) rule, 96
simulated annealing, 116
sink, 12
size
    of AND/OR-network N, 12
    of a cut, 24
    of a hypergraph, 24
slack
    of an AND/OR-network, 21
    of an edge, 21
Smith's rule, 95
solution
    locally optimal, 114
    optimal, 113
    set of solutions, 113
source, 12
successor, 12

direct, 12
tight, 20

taboo search, 116
tail of a directed edge, 23
three field classification scheme, 16
tight, 20
time-cost trade-off curve, 81
total completion time, 14
total weighted completion time, 14

waiting
condition, 10
job, 10

ZUSAMMENFASSUNG

Anfangs eine Problemstellung der Unternehmensplanung und Unternehmensforschung, hat sich Scheduling in den letzten Jahrzehnten als ein wichtiges Teilgebiet der kombinatorischen Optimierung etabliert. Scheduling hat zur Aufgabe einen Zeit- oder Terminplan für eine Menge von Aufgaben, sogenannten Jobs, so aufzustellen, dass eine gegebene Zielfunktion minimiert wird. Im allgemeinen bestehen zwischen den einzelnen Jobs Abhängigkeiten von der Form, dass mit einer Aufgabe erst dann begonnen werden kann, wenn eine bestimmte Menge anderer Jobs bereits abgearbeitet wurde. Wir betrachten in der vorliegenden Arbeit ein verallgemeinertes Modell dieser Vorgängerbeziehungen: AND/OR-Netzwerke. Hierbei erlauben wir außer den eben beschriebenen AND-Abhängigkeiten noch sogenannte OR-Abhängigkeiten, welche besagen, dass ein Job bearbeitet werden kann, sobald mindestens eine Aufgabe aus einer vorher gegebenen Menge von Aufgaben durchgeführt wurde. Anwendung finden diese Arten von Vorgängerbeziehungen beispielsweise in der Compileroptimierung, der Optimierung industrieller Fließbandproduktion und der betriebsmittelbeschränkten Projektplanung.

Diese Arbeit beschäftigt sich mit verschiedenen Fragestellungen des Scheduling mit AND/OR-Netzwerken. In Kapitel 1 wird benötigtes Grundwissen vermittelt und das zugrunde liegende Scheduling-Modell spezifiziert, auf welches die drei folgenden, im wesentlichen voneinander unabhängigen Kapitel zurückgreifen.

Bei der Planung umfangreicher Projekte ist nicht nur die Erstellung eines optimalen Schedules wichtig sondern auch eine nachfolgende Analyse des Schedules in Bezug auf sein Verhalten bei Veränderung der Eingabedaten. Insbesondere die im Voraus nur schätzbaren Vorgangsdauern der einzelnen Jobs unterliegen zufälligen Schwankungen bedingt durch externe Einflüsse wie die Krankheit eines Arbeiters oder Veränderungen in der Beschaffenheit zu bearbeitender Materialien. Die Kenntnis kritischer Jobs, das heißt Aufgaben, deren Vorgangsdauern einen direkten Einfluss auf den Wert der Zielfunktion haben, bietet dem Projektplaner die Möglichkeit, gezielt negativen externen Einflüssen entgegen zu wirken. Der Hauptteil der vorliegenden Arbeit, Kapitel 2, konzentriert sich auf eine solche Untersuchung von Schedules in AND/OR-Netzwerken. Die vorrangige Zielfunktion ist hierbei die Projektdauer, der sogenannte Makespan. Wir definieren vier Varianten von kritischen Jobs und Mengen (von Jobs), mit der Eigenschaft, dass sie den Makespan entweder verkürzen, verlängern oder erhalten bei der Verkürzung be-

ziehungsweise Verlängerung der Vorgangsdauern bestimmter Jobs. Jeweils zwei der Systeme aller Mengen einer Sorte bilden je ein Paar von blockenden Cluttern, welche allerdings nicht die Max-Flow-Min-Cut Eigenschaft besitzen. Wir stellen fest, dass es einfach ist, eine kritische Menge in einem AND/OR-Netzwerk zu finden. Dagegen ist es NP-vollständig zu entscheiden, ob ein bestimmter Job in einer der vier Varianten kritisch ist. Neben verschiedenen strukturellen Charakterisierungen kritischer Mengen beweisen wir außerdem, dass die Veränderung der Vorgangsdauern der verkürzenden und der verlängernden kritischen Mengen exakt an den Makespan weitergegeben wird.

In Kapitel 3 geben wir eine Übersicht über Scheduling AND/OR vorgängerbeschränkter Jobs auf einer oder mehreren identischen Machinen. Im Fall der Makespanminimierung präsentieren wir eine 2-Approximation von Gillies and Liu (1995). Die Minimierung der Summe der gewichteten Abschlusszeiten der Jobs ist wesentlich schwieriger. Eine Reduktion von LABEL COVER zeigt, dass es für jedes $c < 1/2$ NP-schwer ist, diese Zielfunktion mit beliebigen Gewichten auf einen Faktor von $2^{\log^{1-1/\log\log^c n} n}$ des Optimums zu approximieren. Für Einheitsgewichte präsentieren wir einen $\sqrt{n}$-Approximationsalgorithmus, wobei n die Anzahl der Jobs ist. Der gleiche Algorithmus ist eine n-Approximation im Falle von beliebigen Gewichten. Beide Schranken sind scharf.

Im letzten Kapitel der Arbeit betrachten wir eine Relaxierung der AND/OR Bedingungen und die Möglichkeit, mit Verfahren der lokalen Suche ein optimales AND/OR-Netzwerk zu bestimmen. Hierzu geben wir eine kleine Einführung in das Prinzip lokaler Suchverfahren und schlagen eine kanonische Nachbarschaft für AND/OR-Netzwerke vor. Der Beweis, dass unter dieser kanonische Nachbarschaft die Menge aller zulässigen AND/OR-Netzwerke zusammenhängend ist, stellt das Hauptresultat von Kapitel 4 dar. Wir beschließen unsere Untersuchungen mit einer kurzen Diskussion des Job Shop Scheduling Problems und möglichen dadurch inspirierten Strategien für lokale Suchverfahren in AND/OR-Netzwerken.

CURRICULUM VITAE

30. Oktober 1973	Geboren in Konstanz
1980 – 1984	Besuch der Berchen-Grundschule in Konstanz
1984 – 1993	Besuch des Alexander-von-Humboldt-Gymnasiums in Konstanz
1993	Abitur
1993 – 2000	Studium der Mathematik mit Nebenfächern Betriebswirtschaftslehre und Informatik an der Universität Konstanz
1995	Vordiplom in Mathematik mit Nebenfach Betriebswirtschaftslehre
1995/1996	Auslandstudium an der University of Sussex in Brighton, England
1997	Vordiplom im Nebenfach Informatik
2000	Diplom in Mathematik mit Nebenfach Betriebswirtschaftslehre Betreuerin: Prof. Dr. Dorothea Wagner
2000 – 2003	Promotionsstudium Mathematik an der Technischen Universität Berlin als Stipendiatin des Graduiertenkollegs “Combinatorics, Geometry, and Computation” (DFG) Betreuer: Prof. Dr. Rolf H. Möhring
Jul 2001 – Jan 2002	Forschungsaufenthalt bei Prof. Dr. Thomas Erlebach an der Eidgenössischen Technischen Hochschule Zürich, Schweiz