STOCHASTIC RESOURCE-CONSTRAINED PROJECT SCHEDULING

vorgelegt von
Dipl.-Math. techn. Frederik Stork
aus München

Vom Fachbereich Mathematik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften
genehmigte Dissertation

Berichter:	Prof. Dr. Rolf H. Möhring, Technische Universität Berlin
Berichter:	Prof. Dr. Peter Brucker, Universität Osnabrück

Tag der wissenschaftlichen Aussprache: 09. April 2001

Berlin 2001
D 83

PREFACE

After having received my diploma from the Technische Universität Berlin in 1996, Rolf Möhring, the supervisor of my diploma thesis, offered me a research position in his group. At that time I was employed at a Berlin software company the head of which, Gert Scheschonk, strongly encouraged me to accept the offer. I accepted and in 1997 I began to work within a research initiative funded by the Deutsche Forschungsgemeinschaft DFG. The members engaged in this initiative belong to five research groups in Germany which are located at universities in Bonn, Karlsruhe, Kiel, Osnabrück, and Berlin. In Berlin, the scope of the project was to develop algorithms and theory for stochastic resource-constrained project scheduling problems which is the main topic of this thesis.

I am thankful to Rolf Möhring for his support, his encouragement, and the supervision of my thesis. In particular, I greatly benefited from his guidance during my work on AND/OR precedence constraints and scheduling policies.

My special thanks go to my colleagues Martin Skutella and Marc Uetz. Martin greatly helped to establish, generalize, and improve many of my original considerations on AND/OR precedence constraints which finally led to the results presented in Chapters 2 and 3. The continuous fruitful discussion with Marc led to new insights in the field of deterministic resource-constrained project scheduling. The results presented in Chapter 4 on different representations of resource constraints are one example of this productive collaboration.

I am also very grateful to my colleagues Andreas Schulz and Matthias Müller-Hannemann. I gained a lot from Andreas' expertise and his co-authorship in papers on deterministic project scheduling (which are not part of this thesis). My former roommate Matthias was always willing to interrupt his work in order to discuss the questions I raised.

I would also like to mention the fruitful collaboration with the other members of the DFG research initiative on resource-constrained project scheduling. In particular, I thank Peter Brucker for the willingness to serve as a member of my thesis committee.

Some parts of this thesis rely on software implementations that would not have reached the current quality without the support of Ewgenij Gawrilow. I thank him for introducing me to the concept of generic programming; he had a great share in establishing the basis of our programming environment, a collection of fundamental scheduling algorithms and data structures.

Finally, I am grateful to Marc Uetz, Martin Skutella, Andreas Schulz, Marc Pfetsch, Michael Naatz, Ekkehard Köhler, and Andreas Fest for their careful proof-reading of different parts of the manuscript.

It has been a great pleasure to share both research and leisure activities with the colleagues at the Technical University in the groups of Rolf Möhring and Günter Ziegler. It is hard to imagine a better working environment.

Berlin, February 2001 *Frederik Stork*

CONTENTS

INTRODUCTION

Motivation. Scheduling theory is an important and dynamic subject within combinatorial optimization and has attracted numerous researchers. Scheduling is concerned with the planning of activities over time subject to various side constraints with the intention to minimize some objective function. Activities are separate pieces of work and are commonly referred to as *jobs*. In this thesis we consider a fairly general scheduling model that has numerous applications and contains many other models as a special case. Let us sketch the characteristics of the model. First, *precedence constraints* have to be respected, that is, certain jobs must be completed before others can be executed. During its execution, each job requires capacity of different resources, and the resource availability is limited. In addition, we assume that the processing time of each job is uncertain and follows a given probability distribution. The outlined model, which is usually referred to as *stochastic resource-constrained project scheduling*, integrates two different, classical scheduling models both of which have been extensively studied in the past 40 years. On the one hand, this is the deterministic resource-constrained project scheduling problem where each job processing time is assumed to be fixed and known in advance. One of the first papers which refers to this model was written by Wiest (1963). Since then, a vast body of literature has been established; we here only mention the recent publications (Brucker, Drexl, Möhring, Neumann, and Pesch 1999; Węglarz 1999) for reviews of different models and algorithms. On the other hand, stochastic resource-constrained project scheduling generalizes so-called *stochastic project networks* or *PERT-networks* where job processing times are assumed to be stochastic but the resource availability is unlimited. Adlakha and Kulkarni (1989) have provided a bibliography that classifies the enormous number of contributions up to 1987.

Scheduling models with stochastic job processing times are important because many uncertain events within project execution may cause job interruptions and delays. Weather conditions, unavailability of resources, and authorization processes are only some examples. Already Fulkerson (1962) noted that the expected completion of the last job in a stochastic project network, the expected *project makespan*, is greater than or equal to the project makespan that is based on the expected processing times of jobs.

The necessity to consider models with random job processing times is probably best motivated by the following quotation taken from the final report of a re-

cent NASA project (Henrion, Fung, Cheung, Steele, and Basevich 1996). There, space shuttle ground processing is modeled as a stochastic resource-constrained project scheduling problem (we give details in Chapter 6 below).

> "Shuttle ground processing is subject to many uncertainties and delays. These uncertainties arise from many sources, including unexpected shuttle maintenance requirements, failure of ground test equipment, unavailability of resources or technical staff, manifest constraints, and delays in paperwork." (Henrion et al. 1996, Page 4)

Scheduling with policies. Due to the combination of random job processing times and limited resources the stochastic resource-constrained project scheduling problem is a stochastic dynamic optimization problem and, as such, belongs to the field of stochastic dynamic programming. Scheduling is usually done by so-called *policies*. A policy may be seen as a dynamic decision process that defines which jobs are started at certain decision times t, based on the observed past up to t. Since it is commonly believed that the class of all policies is computationally intractable, different subclasses of policies have been considered in the literature. Möhring and Radermacher (1985) have contributed an illustrative survey. Our work is based upon so-called *preselective policies* which have been introduced by Radermacher (1981b). Let us briefly mention the basic concept of this structurally appealing class. Preselective policies are defined via so-called *minimal forbidden sets*. A set F of jobs without a precedence constraint among them is called *forbidden* if the total resource consumption of the jobs in F exceeds the resource availability. If no proper subset of a forbidden set F is forbidden, then we call F *minimal forbidden*. A preselective policy defines for each minimal forbidden set a *preselected* job $j \in F$ which is postponed until at least one job from $F \setminus \{j\}$ has been completed.

Contribution. The purpose of this thesis is to provide new insights on how to solve stochastic resource-constrained project scheduling problems. To this end, we first study the combinatorial structure of preselective policies (and appropriately defined subclasses thereof). Then, we develop, implement, and evaluate solution techniques for stochastic resource-constrained project scheduling problems that are based on these classes of policies. We next outline the contributions in more detail.

The obtained results on the combinatorial structure of preselective policies rely on the concept of so-called *AND/OR precedence constraints* which are a generalization of traditional precedence constraints. For a given set V of jobs, an AND/OR precedence constraint consists of a pair (X, j) with $X \subset V$ and $j \in V \setminus X$ with the meaning that at least one job from X must have been completed before j

can be executed. AND/OR precedence constraints are of relevance in its own due to their appearance within, e. g., assembly or disassembly processes (Goldwasser and Motwani 1999). We propose a new field of application which is based on the fact that any preselective policy can be expressed as a set of such constraints. To this end, we develop a number of basic algorithms for scheduling jobs subject to AND/OR precedence constraints. For example, we give two different polynomial time algorithms to compute earliest job start times as well as a linear time algorithm to detect 'transitive' AND/OR precedence constraints. These algorithms later appear as important components within procedures to compute preselective policies for stochastic resource-constrained project scheduling problems.

The results obtained for AND/OR precedence constraints give also rise to consider particular subclasses of preselective policies. These classes differ with respect to both their computational tractability and the optimum expected objective function value that can be achieved within the respective class. We collect some of the structural and algorithmic properties of these classes of policies and use them to develop in total five branch-and-bound algorithms. Enhanced with many additional ingredients to speed up the computations, the algorithms are rigorously tested on 1440 instances created by the widely accepted instance generator *ProGen* (Kolisch and Sprecher 1996). In particular, for each of the considered classes of policies, we establish results on the trade-off between computational efficiency on the one hand and solution quality on the other hand. In order to deal with the random job processing times we use standard simulation methods, i. e., we generate a set of *scenarios* that mimics the random data.

Finally, to implement the branch-and-bound algorithms, we have to overcome one more difficulty that is related to the forbidden set based definition of preselective policies. Usually, minimal forbidden sets are defined implicitly via resource consumptions of jobs and resource availability. Since we require an explicit representation of the minimal forbidden sets we develop a simple (yet powerful) backtracking algorithm to compute all minimal forbidden sets from a given implicit representation. As a by-product the algorithm suggests a compact representation of all minimal forbidden sets in a tree data structure. This is of particular importance since the number of minimal forbidden sets may be large when compared to the number of jobs.

We already noted above that stochastic resource-constrained project scheduling in general falls into the area of stochastic dynamic programming. However, the focus of our research is the study of preselective policies which are of strong combinatorial structure. As a consequence, the contents of this thesis is rather related to combinatorial optimization than to stochastic dynamic programming.

Outline of this thesis. The thesis is divided into six chapters the first of which provides a brief introduction to the field of *project scheduling*, including different models, basic solutions techniques, and their application.

In *Chapter 2* we study the combinatorial structure of AND/OR precedence constraints. We first discuss the problem of whether a given set of AND/OR precedence constraints is feasible. A set is feasible if there exists an ordering of the jobs in which they can be processed without violating any constraint. We propose a greedy linear-time algorithm that constructs such an ordering if and only if the given set of AND/OR precedence constraints is feasible. Furthermore, we reveal a close relationship to the HORN-SAT problem. We then show that the algorithm to decide feasibility can also be used to find 'transitive' AND/OR precedence constraints. In addition, we prove that there exists a unique minimal representation of a set of AND/OR precedence constraints and suggest a polynomial time algorithm to compute it. We finally argue that the discussed problems are NP-complete for a natural generalization of the model of AND/OR precedence constraints.

In *Chapter 3* we discuss the problem of computing earliest job start times if AND/OR precedence constraints are imposed among jobs. We here consider three different cases: The arc weights of the digraph that can be associated to a given set of AND/OR precedence constraints are (i) arbitrary, (ii) strictly positive, and (iii) non-negative. For Case (i) we show that the problem is closely related to a class of two-person perfect information games played on digraphs (so-called mean-payoff games). Moreover, we extend the feasibility criterion established in the previous chapter, which immediately implies that the decision problem is in NP $\cap$ co-NP. In fact this result was known for mean-payoff games. We then give polynomial time algorithms for Cases (ii) and (iii). Finally, we study the *time-cost tradeoff problem* for the case of AND/OR precedence constraints. In the time-cost tradeoff problem, the jobs share a common resource which can be freely distributed among them, but which affects their processing times. This dependency yields a tradeoff between an early project completion and a low project cost. We discuss the analytical properties of the time-cost tradeoff curve and show that it is NP-hard to compute a single point of the curve.

In *Chapter 4* we study alternative representations of resource constraints. Traditionally, resource constraints are defined via a set K of different resources. Each job $j \in V$ requires capacity of r_{jk} units of resource $k \in K$ while being processed, and the total resource availability R_k of each resource $k \in K$ is limited. Let us call this representation of resource constraints the *threshold representation*. Another representation is by the previously mentioned minimal forbidden sets. We first discuss the computational complexity of different problems that are related to both the threshold representation of resource constraints and the representation by minimal forbidden sets. We identify an interesting relation to so-called threshold

(hyper-)graphs which shows that the problem of finding the minimal number of resource types k required in a threshold representation for a given system of minimal forbidden sets is NP-complete. If resource constraints are given in threshold representation, we show that, given a job $j \in V$, it is NP-complete to decide whether there exists a minimal forbidden set F with $j \in F$. We propose a backtracking algorithm which computes the system $\mathcal{F}$ of minimal forbidden sets for an instance which is given by the usual threshold representation. We show that the algorithm can be implemented to run in polynomial time with respect to the in- and output for instances with only one resource type (that is, $|K| = 1$). The algorithm immediately suggests a tree-like data structure to efficiently represent all minimal forbidden sets. We finally report on a computational evaluation of the algorithm. The results exhibit the benefits of the proposed algorithm in comparison to a previously suggested approach to compute all minimal forbidden sets by Bartusch (1984).

In *Chapter 5* we study the class of *preselective policies* as well as different subclasses thereof. We first review Radermacher's notion of a policy and discuss popular classes of policies (so-called *priority policies* and *Earliest Start policies*). We then show that preselective policies can be expressed by a set of AND/OR precedence constraints and, with the results presented in Chapter 2, we establish a necessary and sufficient dominance criterion. In addition, we derive an efficient algorithm that (approximately) computes the expected objective function value that results from a project execution according to a specific preselective policy. The algorithm is based on the results of Chapter 3. We then introduce a new subclass of preselective policies, so-called *linear preselective policies*. The essential advantage is that algorithms for such policies operate on acyclic structures, which allows a simpler and more efficient computational handling. Next, we study another subclass of preselective policies, the so-called *job-based priority policies*. In contrast to (linear) preselective policies and ES-policies, they have the important advantage that the representation of resource constraints by (possibly exponentially many) minimal forbidden sets is not required since the threshold representation suffices. As a consequence, algorithms that are based on job-based priority policies have the potential to be applicable to projects where a large number of minimal forbidden sets makes the use of forbidden set based policies computational inefficient. Finally, we relate the optimum expected objective function values that can be achieved within the above mentioned classes of policies to each other.

The last chapter, *Chapter 6*, is concerned with the computational evaluation of the theoretical results presented in Chapter 5. The objective function that we consider is the minimization of the expected project makespan. We establish results on the trade-off between computational efficiency on the one hand and solu-

tion quality on the other hand, for the classes of policies discussed in Chapter 5. We utilize two different branching schemes as well as several additional ingredients such as dominance rules and lower bounds to speed up the computations. In total, we have implemented two different branch-and-bound algorithms for linear preselective policies and one algorithm for preselective policies, ES-policies, and job-based priority policies, respectively. We explore their computational efficiency on 1440 instances of different size. The experiments reveal that linear preselective polices are computationally more tractable than preselective policies and ES-polices. For projects that involve a moderate number of minimal forbidden sets it is possible to compute (near) optimal linear preselective policies with truncated versions of the branch-and-bound algorithm. In addition, the experiments exhibit that the optimum expected makespan among the class of job-based priority policies is only slightly larger compared to the optimum makespan within the class of preselective policies. As a consequence, since job-based priority policies do not require the forbidden set representation of resource constraints, they are a good starting point to develop heuristic approaches to solve large-scaled stochastic resource-constrained project scheduling problems.

We assume that the reader is familiar with the fundamental concepts of combinatorial optimization which can be found, e. g., in the books of Papadimitriou and Steiglitz (1982), Cook, Cunnigham, Pulleyblank, and Schrijver (1998), and Korte and Vygen (2000). Other books on combinatorial optimization have a more specific focus, for instance, flows in networks (Ahuja, Magnanti, and Orlin 1993) or integer linear programming (Schrijver 1986). A comprehensive treatment of complexity theory is given, e. g., in the books of Garey and Johnson (1979) and Papadimitriou (1994). There are several books and survey articles on scheduling theory with different points of emphasis. We want to mention the books of Brucker (1998) and Pinedo (1995) and the surveys of Lawler, Lenstra, Rinnooy Kan, and Shmoys (1993) and Brucker, Drexl, Möhring, Neumann, and Pesch (1999). The prerequisites of probability theory are relatively few because we use standard simulation methods in order to deal with (arbitrary) job processing time distributions. Finally, we sometimes make use of order-theoretic notation; for an introduction to the theory of partially ordered sets we refer to (Trotter 1992).

Parts of this thesis have been published or pre-published in (Möhring, Skutella, and Stork 2000a; Möhring, Skutella, and Stork 2000b; Stork and Uetz 2000; Möhring and Stork 2000; Stork 2000).

CHAPTER 1

PROJECT SCHEDULING: COMPLEXITY AND BASIC SOLUTION TECHNIQUES

The purpose of this brief introductory chapter is to classify the model of stochastic resource-constrained project scheduling within the field of combinatorial optimization. We review the complexity status and classical solution techniques of some popular special cases. These are the *deterministic resource-constrained project scheduling problem* and *stochastic project networks* or *PERT-networks*. We also (briefly) touch the field of *stochastic machine scheduling*.

1.1 Deterministic Resource-Constrained Project Scheduling

Problem definition. We first consider the deterministic resource-constrained project scheduling problem. An instance (or *project*) consists of a finite set $V = \{1, \ldots, n\}$ of jobs together with a *partial order* $G_0 = (V, E_0)$, $E_0 \subset V \times V$, on the set of jobs. In order to perform a project, all jobs of V have to be executed in accordance with the precedence constraints that are defined by the partial order: If $(i, j) \in E_0$ then j cannot be started before the completion of i. In addition to that, jobs need different (renewable) resources $k \in K$ while being processed. A constant amount of $R_k \in \mathbb{N}$ units of each resource is available throughout the project and each job j requires $0 \leqslant r_{jk} \leqslant R_k$ ($r_{jk} \in \mathbb{N}$) units of resource $k \in K$ while in process. The processing time $p_j \in \mathbb{R}_>$ of each job $j \in V$ is deterministic and known in advance ($\mathbb{R}_>$ denotes the set of positive real numbers). Moreover, it is assumed that each job is executed *non-preemptively*, that is, the execution of jobs must not be interrupted. A *schedule* is a vector $S \in \mathbb{R}^n_\geqslant$ of job start times $S_j \in \mathbb{R}_\geqslant$, $j \in V$ ($\mathbb{R}_\geqslant$ denotes the set of non-negative real numbers). S is called *time-feasible* if S respects all precedence constraints, i. e., $S_j \geqslant S_i + p_i$ for all $(i, j) \in E_0$. S is called *resource-feasible* if, at any time t and for each resource k, the sum of the resource consumption of all jobs which are in process at t does not exceed the availability R_k. Together, we call a schedule *feasible* if it is both time- and resource-feasible. For a given schedule S, the *completion time* C_j of job $j \in V$ is defined as $S_j + p_j$. The objective is to find a feasible schedule such that a given measure of performance $\kappa : \mathbb{R}^n_\geqslant \to \mathbb{R}_\geqslant$ which maps a vector of comple-

tion times to a (non-negative) real value, is minimized. Throughout the thesis we assume that κ is *regular*, that is, κ is non-decreasing. Many of the popular performance measures have this property, e. g., the *makespan* $C_{max} := max_{j \in V} C_j$ and the *weighted sum of completion times* $\sum_{j \in V} w_j C_j$ (the weight $w_j \geqslant 0$ indicates the importance of jobs). In the sequel, we call κ the *cost function* of the project and refer to the value $\kappa(C)$ with $C = (C_1, \ldots, C_n)$ as the *project cost*.

We next briefly review previous work on deterministic resource-constrained project scheduling. In fact, since the number of contributions is enormous, we only consider selected topics that are of relevance for this thesis. We refer to (Brucker, Drexl, Möhring, Neumann, and Pesch 1999) and (Węglarz 1999) for surveys on different models, recent research directions as well as many references.

Complexity. As a generalization of many classical NP-hard machine scheduling problems, the resource-constrained project scheduling problem is also NP-hard (in the strong sense). But even more, it is among the most intractable combinatorial optimization problems, as is perhaps best underlined by the fact that the vertex coloring problem in graphs can be expressed as a special case of a resource-constrained project scheduling problem with makespan objective. The following transformation is described in, e. g., (Schäffter 1997). For an instance of vertex coloring, introduce a job j with unit processing time $p_j = 1$ for each vertex. Then, add a resource k for each edge, let $R_k = 1$ and set $r_{jk} = 1$ for the two jobs which are incident to the edge, and $r_{jk} = 0$, otherwise. The makespan of the so-constructed scheduling instance equals the minimal number of colors required to color the vertices of the graph. Thus, as for vertex coloring, there is no polynomial-time approximation algorithm with a performance guarantee less than n^{ϵ} for some $\epsilon > 0$, unless $P = NP$ (Feige and Kilian 1998).

Exact procedures. A number of branch-and-bound procedures which compute an optimal solution to the problem has been considered in the literature. Interestingly, they are based upon quite different ideas on how the enumeration tree which represents all feasible solutions is organized. A popular approach is to start with the earliest start schedule (with respect to the precedence constraints) and then systematically postpone sets of jobs (so-called *delaying alternatives*) in order to obtain feasible schedules. In different variations this principle is used by, e. g., Stinson, Davis, and Khumawala (1978), Demeulemeester and Herroelen (1992, 1997), and Mingozzi, Maniezzo, Ricciardelli, and Bianco (1998). A constrained-programming-based enumeration is studied by Dorndorf, Pesch, and Phan Huy (2000a, 2000b): Starting from an interval of feasible start times for each job, the branching process reduces the size of the intervals until all intervals consist of only one value. Another approach is described in (Brucker, Knust, Schoo, and

Thiele 1998). They systematically partition the set $V \times V$ of pairs (i, j) into three sets which indicate the position of i and j relative to each other. For each such triplet of sets a component-wise minimal schedule which satisfies the conditions imposed by the triplet can be computed in polynomial time (if one exists). The branching process iteratively assigns each job pair to one of the three groups such that finally, all triplets (which represent at least one feasible schedule) are enumerated. There are two other approaches, we call them the precedence-tree branching scheme and the forbidden set branching scheme. The schemes, which are described by Patterson, Słowiński, Talbot, and Węglarz (1989) and Igelmund and Radermacher (1983a), respectively, are discussed in detail in Chapter 6 below.

Lower bounds. There are numerous publications which deal with the computation of lower bounds on the minimal project makespan for resource-constrained projects; we briefly review some of them. Various extensions of critical path analysis have been suggested, and perhaps the first reference in this direction is by Stinson, Davis, and Khumawala (1978). Linear programming lower bounds are analyzed by Christofides, Alvarez-Valdes, and Tamarit (1987), as well as Cavalcante, de Souza, Savelsbergh, Wang, and Wolsey (1998) and Möhring, Schulz, Stork, and Uetz (2000). Based on another integer programming formulation, several linear programming lower bounds are studied by Mingozzi, Maniezzo, Ricciardelli, and Bianco (1998). Klein and Scholl (1999) propose a destructive improvement approach, which is based on the idea to reject fictitious upper bounds by proving infeasibility, just like in constraint propagation. Similar techniques are used by Heilmann and Schwindt (1997). Based on the formulation by Mingozzi et al. (1998) and combined with constraint propagation techniques, Brucker and Knust (2000) obtain the currently best known lower bounds on a widely used benchmark test set. Finally, a Lagrangian relaxation based approach is suggested by Christofides, Alvarez-Valdes, and Tamarit (1987). They solve the resulting Lagrangian subproblem by branch-and-bound. Möhring, Schulz, Stork, and Uetz (2000) reconsider the same Lagrangian relaxation and use a maximum-flow algorithm to solve the subproblem.

Feasible solutions. Most relevant from a practical point of view is the computation of feasible solutions (schedules). Not surprisingly, this topic has been studied most frequently. Besides classical list scheduling heuristics, almost every local search technique has been applied to the problem (with makespan objective, among others). Hartmann and Kolisch (1998) review and evaluate some of the heuristics. This includes a genetic algorithm by Hartmann (1999), a simulated annealing algorithm by Bouleimen and Lecocq (2000), as well as sampling-based list-scheduling heuristics by Kolisch (1996). Other solution procedures are

based on (truncated) branch-and-bound algorithms, e. g., (Sprecher 2000; Dorndorf, Pesch, and Phan Huy 2000a). Möhring, Schulz, Stork, and Uetz (2000) study list scheduling heuristics that use dual information from solutions of the earlier mentioned Lagrangian relaxation.

Test sets. Most of the above mentioned approaches have been implemented and tested on different sets of benchmark instances that have been created over the years. Many of the evaluations before 1996 have been performed on a test set with 110 instances that were (randomly) generated by Patterson (1984). The sizes of these instances, i. e., the number n of jobs, varies from 7 to 51 (26 on average). Nowadays, solutions with minimum makespan can be computed for each of the 110 instances within very short time by branch-and-bound procedures, see, e. g., (Demeulemeester and Herroelen 1992). In order to test algorithms on larger and harder instances, Kolisch and Sprecher (1996) develop a generator called *ProGen* which allows to create instances of different size and with different characteristics. They established the benchmark library PSPLIB (2000) which currently contains instances with 30, 60, 90, and 120 jobs. In total, they created 480 instances for each of the instance sizes 30, 60, and 90, and 600 instances with 120 jobs. Currently, state-of-the-art branch-and-bound algorithms can solve each of the instances with 30 jobs optimally, however, already for many instances with 60 jobs, the minimal makespan is unknown (124 out of 480 instances). We give more details on the test sets in Chapters 4 and 6 below.

1.2 Stochastic Project Networks (PERT-Networks)

As with the resource-constrained project scheduling problem, a stochastic project network consists of a set V of jobs with precedence constraints among pairs of jobs. However, in contrast to the previously discussed problems, no resource constraints are imposed. As a consequence, each job can be started as soon as all its predecessors in the partial order defined by the precedence constraints are completed. The difficulty arises from the additional assumption that the processing time of each job is uncertain and is given by a random variable $\boldsymbol{p}_j$ (as a notational convention, throughout the thesis, we always use bold letters to denote a random variable). For a given stochastic project network the objective is to determine the *project makespan distribution* or some characteristic thereof, e. g., its expectation.

In the context of stochastic project networks, jobs are often called *activities*. The set of precedence constraints is represented as a directed acyclic graph where each job is identified with an arc of the digraph (*activity-on-arc* network). In the literature, stochastic project networks are often referred to as *PERT-networks*,

since PERT was one of the first techniques to analyze the stochastic behavior of such networks. Malcom, Roseboom, Clark, and Fazar (1959) originally introduced the term PERT as an abbreviation for *Program Evaluation Research Task* which was later renamed to *Project Evaluation and Review Technique*. Adlakha and Kulkarni (1989) established a classified bibliography of the vast body of literature up to 1987.

We introduce the following notation. Let $\boldsymbol{p} = (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n)$ be the vector of job processing times and $p = (p_1, \ldots, p_n) \in \mathbb{R}^n_>$ be a particular *scenario* drawn from $\boldsymbol{p}$. Then, $S(\boldsymbol{p})$ and $S(p)$ is the vector of random start times and the vector of start times according to the scenario p, respectively. We equivalently define $C(\boldsymbol{p}) = S(\boldsymbol{p}) + \boldsymbol{p}$ and $C(p) = S(p) + p$ to be the vector of expected completion times and the vector of completion times according to the scenario p, respectively. Now, since the project makespan distribution is the maximum of the lengths of each chain Q of G_0, it can formally be written as $C_{\max}(\boldsymbol{p}) := \max_{Q \in \mathcal{Q}} \sum_{j \in Q} \boldsymbol{p}_j$. Here, $\mathcal{Q}$ denotes the set of (maximal) chains of G_0.

Complexity. We next assume that all processing times are independent, discrete random variables that may take two values. For a given deadline $T \geqslant 0$ define DF (=Distribution Function) to be the problem to determine $Prob(C_{\max}(\boldsymbol{p}) \leqslant T)$. It is maybe a surprising fact that DF is a #P-complete problem. The difficulty stems from the fact that the (random) lengths of the chains are dependent in general, even if job processing time distributions are independent (jobs usually belong to more than one chain). Hagstrom (1988) established the result by adapting the work of Provan and Ball (1983) on network reliability problems. The network reliability problem is as follows. Given a directed acyclic graph D with a source a and a sink b and a failure probability q_j for each arc j, determine the probability that D is *reliable*, i. e., there exists a path between a and b without arc failures. We consider the special case that each path in D consists of three arcs and that the failure probabilities are identical, i. e., $q_j = q$. The work of Provan and Ball (1983) includes the fact that computing the probability that a so-structured digraph D is reliable is a #P-complete problem. For a given instance of this network reliability problem we construct a stochastic project network as follows. The digraph D is interpreted as an activity-on-arc network which defines the set of jobs (one job for each arc) and the set of precedence constraints. The (random) processing time of each job j is 0 with probability q and 1 with probability $1-q$. Then, the probability that D is reliable equals 1 minus the probability that the project makespan is less than or equal to 2. This transformation as well as a proof that DF is in #P is due to Hagstrom (1988).

However, if the given network is *series-parallel*, the problem can be solved more efficiently. In fact, in this case all required computations can be performed

in such a way that the involved distributions are independent. Based on this observation, various solution procedures have been suggested to heuristically compute or bound the makespan distribution, e. g., (Martin 1965; Kleindorfer 1971; Dodin 1985). It should be noted, however, that the problem DF is still NP-hard in the weak sense as is shown by Möhring and Müller (1998), see also (Möhring 2000a).

Simulation. A widely used technique to heuristically compute the project makespan distribution (or some characteristic thereof) is *simulation*. The basic methodology (adapted to our application) is as follows: Iteratively generate a scenario p from $\boldsymbol{p}$. Then the makespan for each such scenario p is determined by a standard longest path computation and the resulting value $C_{max}(p)$ together with the probability of p is stored. After a 'sufficiently large' number of iterations has been performed, standard statistical methods can be used to estimate the distribution of $C_{max}(\boldsymbol{p})$. Van Slyke (1963) was among the first who applied this framework to stochastic project networks; the methodology and other references are collected in (Adlakha and Kulkarni 1989, Cathegory IV). For the concept of simulation in general and other fields of applications see, e. g., (Bratley, Fox, and Schrage 1987).

1.3 Stochastic Resource-Constrained Project Scheduling

Problem definition. Stochastic resource-constrained project scheduling combines the features of the two previously described models. A set of jobs with random processing times have to be executed subject to both precedence and resource constraints. There is, however, an important new aspect that comes into play: What is a *solution* to a stochastic resource-constrained project scheduling problem? Neither a schedule nor the project cost distribution contains enough information to make decisions during the execution of the project. Moreover, the project cost distribution actually depends on the decision that are made during project execution. Hence, a solution should define for each possible 'event' that appears within the execution of the project an appropriate 'action'. This 'action' typically is a decision which jobs should be executed next and an 'event' may be the completion of some jobs. To make such a decision one may want to exploit the information given by the current state of the project. Such a solution is called a *policy*. For the moment, we leave the reader with this intuition; we provide a formal definition of a policy as well as many details on particular classes of such policies in Chapter 5 below.

Let us assume that we execute a given project according to some given policy Π. Once the project is finished we know the processing time p_j and the completion time C_j of each job $j \in V$. For a given cost function κ we can now compute the

cost $\kappa(C)$ of the project. Clearly, the completion times $C = C^{\Pi}(\boldsymbol{p})$ are dependent on the chosen policy Π and the random job processing times $\boldsymbol{p}$. In particular, $C^{\Pi}(\boldsymbol{p})$ and also the project cost $\kappa(C^{\Pi}(\boldsymbol{p}))$ is a random variable. The objective is to optimize some characteristic of the corresponding distribution. In the thesis, the objective is to minimize the project costs *in expectation*. Using the notation of Sections 1.1 and 1.2, the stochastic resource-constrained project scheduling problem can now be summarized as follows.

> STOCHASTIC RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEM: Let V be a set of jobs with random processing times $\boldsymbol{p}$ and let E_0 be a set of precedence constraints. Moreover, let K be a set of resources with resource availabilities R_k, $k \in K$, and resource consumptions r_{jk}, $j \in V$, $k \in K$. For a given regular cost function κ, find a policy Π that minimizes κ in expectation and compute the expected project costs $\kappa(C^{\Pi}(\boldsymbol{p}))$.

For illustration, we will often use the following example of a stochastic resource-constrained project scheduling problem. Except for the processing time distributions the example is taken from (Igelmund and Radermacher 1983b).

Example 1.3.1. *Let $G_0 = (V, E_0)$ be given by $V = \{1, 2, 3, 4, 5\}$ and $E_0 = \{(1, 4), (3, 5)\}$. There are two resources with availability $R_1 = 1$ and $R_2 = 2$ and the resource consumptions of jobs are defined as $r_{1,1} = r_{5,1} = r_{2,2} = r_{3,2} = r_{4,2} = r_{5,2} = 1$ and $r_{jk} = 0$, otherwise. Furthermore, expected job processing times are $E[\boldsymbol{p}] = (3, 5, 3, 5, 6)$. The random variables $\boldsymbol{p}_j$ are independent and uniformly distributed with variance 2 and the objective is to minimize the expected project makespan.*

Representation of processing time distributions and complexity. In principle, to construct an instance of the above defined stochastic resource-constrained project scheduling problem, one requires sufficient information to derive a complete probability distribution for each job processing time. From a practical point of view, this is certainly a rather unrealistic assumption. However, one may estimate each job's expected processing time as well as its variance, for instance, based on historical data. Then, together with some typical type of distribution (e. g., a uniform, normal, or exponential distribution), one can state the desired probability distributions. Models that need fewer information on job processing times have also been considered in the literature. For instance, in (Sotskov, Tanaev, and Werner 1998; Bast 1998) it is assumed that only a lower and an upper bound on each job processing time is known.

We next discuss the complexity status of the stochastic resource-constrained project scheduling problem. Motivated by the above considerations, throughout

the thesis we assume that each processing time distribution $\boldsymbol{p}_j$ is encoded by a constant number of integers. This can be, e. g., an integer for the mean and the variance, plus information on the type of the distribution (as outlined above). Alternatively, an arbitrary continuous distribution can be represented by a piecewise linear approximation with a constant number of supporting points. It then follows from the remarks on the deterministic resource-constrained project scheduling problem that the stochastic counterpart is NP-hard as well.

Moreover, job processing times p_j might in principle be (positive) real numbers. However, in the context of complexity theoretic matters (such as worst case running times of algorithms), we assume that each job processing time p_j is represented by a rational number. As a consequence, each p_j has a succinct encoding and every basic operation can be executed in $\mathrm{O}(\,1\,)$ time.

Related work. The relevant literature on stochastic resource-constrained project scheduling can be grouped into two categories. On the one hand, these are theoretical studies on general or particular classes of policies, and, on the other hand, their computational application and evaluation. We give an elaborate review on previous work in both categories in Chapters 5 and 6, respectively. Let us however, list the relevant references here. The definition of a policy (in the form we use it) is due to Radermacher (1981b). A number of results that are related to special classes of policies were established subsequently (Radermacher 1981a; Igelmund and Radermacher 1983b; Möhring, Radermacher, and Weiss 1984; Möhring, Radermacher, and Weiss 1985; Möhring and Radermacher 1985; Radermacher 1986). Independently from the above mentioned work, scheduling policies are also studied by Fernandez and Armacost (1996) and Fernandez, Armacost, and Pet-Edwards (1998a, 1998b). There are only few computational publications on the stochastic resource-constrained project scheduling problem. Igelmund and Radermacher (1983a) report on experiments with a branch-and-bound algorithm. Since then, Golenko-Ginzburg and Gonik (1997), Tsai and Gemmill (1998), and Valls, Laguna, Lino, Pérez, and Quintanilla (1998) considered the problem; they develop greedy and local search heuristics. Finally, the NASA (Henrion et al. 1996) has set up a research project to improve efficiency within Space Shuttle ground processing.

Stochastic machine scheduling. Various *stochastic machine scheduling* problems that have been considered in the literature can be expressed as a special case of the stochastic resource-constrained project scheduling problem. As an example, parallel identical machines can be modeled by setting $|K| = 1$ and $r_{j1} = 1$ for each job j. There exist numerous contributions in which it is shown that a particular policy is optimal for some particular machine scheduling setting.

Unfortunately, the used techniques to establish these results do not generalize to resource-constrained project scheduling problems. However, one interesting topic should be mentioned. There exist deterministic machine scheduling problems that are NP-hard while a stochastic counterpart is solvable in polynomial time. As an example, consider the deterministic problem where jobs have to be assigned to m parallel identical machines with the objective to minimize the makespan. This problem is well known to be NP-hard. However, if the processing times of jobs are independent and exponentially distributed it is an optimal policy to always start a job with longest expected processing time whenever a machine becomes available (Bruno, Downey, and Frederickson 1981). Notice that results of that type usually do not include the computation of the expected cost of the respective policy in polynomial time.

For more information on models, results, and references in stochastic machine scheduling and also stochastic shop scheduling, we refer to the book of Pinedo (1995, Part 2), to the survey (Lawler, Lenstra, Rinnooy Kan, and Shmoys 1993, Section 16), as well as the book chapter (Righter 1994).

CHAPTER 2

AND/OR PRECEDENCE CONSTRAINTS: STRUCTURAL ISSUES

In many scheduling applications it is required that the processing of some job must be postponed until some other job, which can be chosen from a pre-given set of alternatives, has been completed. The traditional concept of precedence constraints fails to model such restrictions. Therefore, the concept has been generalized to so-called *AND/OR precedence constraints*. The model is of relevance within the context of stochastic resource-constrained project scheduling because, as we show later, it can be used to represent preselective policies.

Notice that neither resource limitations nor stochastic job processing times are considered in the chapter. The material is taken from a joint publication with Rolf Möhring and Martin Skutella (Möhring, Skutella, and Stork 2000b).

2.1 Motivation and Related Work

For a given set V of jobs, a traditional precedence relation usually comprehends the requirement that a job j cannot be started before another job i has been completed since, e. g., the execution of j requires (parts of) the output of job i. In this case, precedence constraints are given by a set E_0 of ordered pairs (i, j), $i \neq j \in V$, inducing a partially ordered set $G_0 = (V, E_0)$. In a feasible implementation of the project, the jobs have to be executed in accordance with G_0. Since, in this setting, each job j can only start after the completion of *all* its direct predecessors in G_0, we call these precedence constraints AND-*constraints*. (A job i is a direct predecessor of j with respect to G_0 if there exists a non-transitive pair $(i, j) \in E_0$.)

However, there are many applications where a job can be executed as soon as *any* of its direct predecessors has been completed; we refer to such temporal restrictions as OR-*constraints*. Traditional precedence constraints fail to model this requirement. Consequently, the model has been generalized to so-called AND/OR *precedence constraints* which can be represented by a set $\mathcal{W}$ of pairs (X, j) with the meaning that job $j \in V$ cannot be executed before some job $i \in X \subseteq (V \backslash \{j\})$ has been completed. Alternatively to the term AND/OR precedence constraint we

call a pair (X, j) a *waiting condition*. Moreover, we call X the *predecessor set*, and job j the *waiting job* of (X, j). Notice that for a singleton $X = \{i\}$, the constraint (X, j) is a traditional AND-constraint (i, j).

An intuitive motivation for AND/OR precedence constraints is noted in (Gillies and Liu 1995). An engine head has to be fixed by four bolts. However, one of the bolts may secure the engine head well enough to allow further work on it. If the set X consists of the four jobs to secure the bolts and j represents the further work on the engine head, then the waiting condition (X, j) obviously models the desired temporal dependencies among the jobs. Another motivation is studied by Goldwasser and Motwani (1999). They consider the problem of partially disassembling a given product to reach a single part (or component). In order to remove a certain part, one previously may have to remove other parts which can be modeled by traditional (AND) precedence constraints. However, one may choose to remove that same part of the product from another geometric direction, in which case some other parts must be removed previously. This freedom of choice can be modeled by AND/OR precedence constraints.

The combinatorial structure of a system $\mathcal{W}$ of waiting conditions occurs in different fields of discrete mathematics and theoretical computer science. In the context of *directed hypergraphs* each $(X, j) \in \mathcal{W}$ represents a hyperarc with a set X of source nodes and a single target node j. Ausiello, d'Atri, and Saccà (1983) (see also Ausiello, d'Atri, and Saccà (1986)) generalize transitive closure and reduction algorithms from directed graphs to directed hypergraphs. Another related class of combinatorial objects are *antimatroids* (special greedoids) which can be defined via a set of waiting conditions, see, e. g., (Korte, Lovász, and Schrader 1991, Page 22). Furthermore, many problems stemming from artificial intelligence can be formulated by hierarchies of subproblems where different alternatives exist to solve these subproblems, see, e. g., (Nilsson 1980). There, a graphical representation of such hierarchies is called AND/OR graph.

In the context of scheduling, Goldwasser and Motwani (1999) derive inapproximability results for two particular single-machine scheduling problems with AND/OR precedence constraints. Gillies and Liu (1995) consider single and parallel machine scheduling problems with different structures of AND/OR precedence constraints; they prove NP-completeness of finding feasible schedules in some settings that are polynomially solvable with traditional precedence constraints. Moreover, they give approximation algorithms for some makespan minimization problems.

For AND-constraints, fundamental problems such as deciding feasibility, finding transitive AND-constraints, and computing earliest start times of jobs can be solved efficiently by applying classical graph algorithms. Most important for the algorithmic treatment is the fact that AND-constraints define acyclic structures on the set V of jobs such that many problems can be solved by considering jobs in

the order of a topological sort. Since this is generally not the case for AND/OR precedence constraints, the algorithms for AND-constraints cannot be applied in that setting. In this chapter we introduce efficient algorithms and structural results for the more general and complex model of AND/OR precedence constraints. We show that feasibility as well as questions related to generalized transitivity can be solved by applying essentially the same linear-time algorithm. Moreover, we discuss a natural generalization of AND/OR precedence constraints and prove that the same problems become NP-complete in this setting. The topic of computing earliest job start times for given temporal distances between jobs is deferred to Chapter 3.

The chapter is organized as follows. After stating some basic requirements in Section 2.2, we discuss feasibility of a set of AND/OR precedence constraints and a relation to the HORN-SAT problem in Section 2.3. Problems that are related to generalized transitive closure and reduction are considered in the Sections 2.4 and 2.5. Finally, the above mentioned generalization of AND/OR precedence constraints is studied in Section 2.6.

2.2 Preliminaries

In order to illustrate the presentation in this chapter as well as in Chapter 3 we use the following example.

Example 2.2.1. *Let* $V := \{1, \ldots, 7\}$ *be the set of jobs and* $\mathcal{W} := \{w_1, w_2, w_3, w_4, w_5\}$ *be the set of waiting conditions where* $w_1 = (\{1, 5\}, 4)$*,* $w_2 = (\{2, 6\}, 4)$*,* $w_3 = (\{4, 3\}, 6)$*,* $w_4 = (\{4\}, 5)$*, and* $w_5 = (\{4, 5, 6\}, 7)$*.*

Graph-representation. We use a natural representation of AND/OR precedence constraints by a directed graph D on the set $\mathcal{V} = V \cup \mathcal{W}$ of nodes. Thus, D has one node for each job and one node for each waiting condition. The set A of arcs is constructed in the following way: For every waiting condition $w = (X, j) \in \mathcal{W}$, we introduce arcs (i, w), for each $i \in X$, and one additional arc (w, j). The size of the resulting digraph D is linear in the input size of the problem. The sets V and $\mathcal{W}$ form a bipartition of D. Similar digraphs are used to represent directed hypergraphs, see, e. g., (Gallo, Longo, Pallottino, and Nguyen 1993) and (Gallo, Gentile, Pretolani, and Rago 1998). In general, D may contain cycles. For a node $j \in V \cup \mathcal{W}$, we use $in(j)$ and $out(j)$ to denote the sets $\{i \in V \cup \mathcal{W} | (i, j) \in A\}$ and $\{i \in V \cup \mathcal{W} | (j, i) \in A\}$, respectively. We also sometimes use the notation $in_D(j)$ and $out_D(j)$ to stress the underlying digraph D.

The digraph resulting from Example 2.2.1 is depicted in Figure 2.1. For the moment, the numbers associated with the arcs can be ignored; they come into

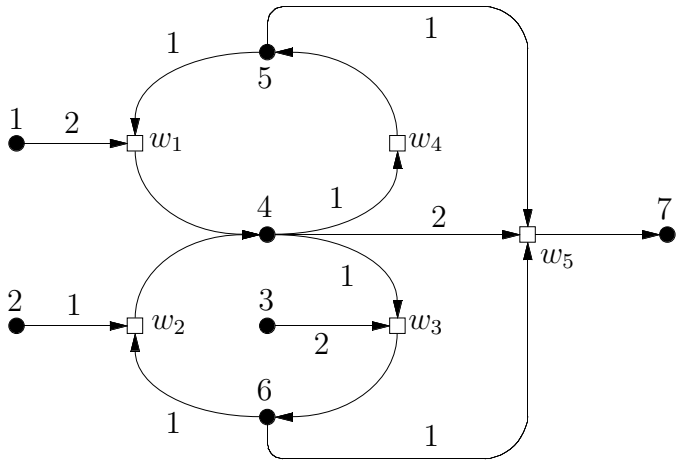

Figure 2.1: The digraph resulting from Example 2.2.1. Circular nodes correspond to jobs (AND-nodes) while square nodes represent waiting conditions (OR-nodes). Numbers associated with arcs define time lags used in Chapter 3 below (the time lags of arcs without a number are 0).

play in Chapter 3 when earliest job start times are computed. As usual, a cycle in D is a sequence $(v_0, v_1, v_2, \ldots, v_k, v_0)$, $v_\ell \in \mathcal{V}$, where $(v_0, v_1, v_2, \ldots, v_k)$ is a directed path and there exists an arc from v_k to v_0. We also consider *generalized cycles* which are induced subgraphs D' of D that consist of node sets $V' \subseteq V$ and $\mathcal{W}' \subseteq \mathcal{W}$ such that $in_D(j) \cap \mathcal{W}' \neq \emptyset$ for each $j \in V'$ and $\emptyset \neq in_D(w) \subseteq V'$ for each $w \in \mathcal{W}'$.

Realizations. Given a set V of jobs and a set $\mathcal{W}$ of waiting conditions, an implementation of the corresponding project requires a decision for each waiting condition (X, j): One has to determine a job $i \in X$ job j should wait for. The entirety of these decisions, i. e., a set E of ordered pairs must define a partial order $R = (V, E)$ on the set V of jobs (the introduction of a cycle would lead to infeasibility) such that,

$$\text{for each } (X, j) \in \mathcal{W}, \text{ there exists an } i \in X \text{ with } (i, j) \in E. \qquad (*)$$

Conversely, every partial order R with property (*) defines an implementation of the project and is therefore called a *realization* for the set $\mathcal{W}$ of waiting conditions. In what follows, a set $\mathcal{W}$ of waiting conditions is called *feasible* if and only if there exists a realization for $\mathcal{W}$. If a partial order $R' = (V, E')$ is an extension of R, i. e., E is a subset of E', R' is also a realization because it obviously fulfills property (*). In particular, a set of AND/OR precedence constraints is feasible if

Algorithm 1: Feasibility check of a set of waiting conditions

Input : A set V of jobs and waiting conditions $\mathcal{W}$.
Output: A list L of jobs from V.

$Q := \emptyset$; $L := \emptyset$;
for *jobs* $j \in V$ **do**
    $a(j) := |\{(X, j) \in \mathcal{W}\}|$;
    if $a(j) = 0$ **then** add j to Q;
while $Q \neq \emptyset$ **do**
    remove a job i from Q;
    insert i at the end of L;
1   **for** *waiting conditions* $(X, j) \in \mathcal{W}$ *with* $i \in X$ **do**
        decrease $a(j)$ by 1;
        if $a(j) = 0$ **then** add j to Q;
        remove (X, j) from $\mathcal{W}$;
return L;

and only if there exists a total order of the jobs which is a realization; we call such a realization *linear*.

Possible linear realizations of Example 2.2.1 are for instance $1 \prec \ldots \prec 7$ and $3 \prec 6 \prec 7 \prec 2 \prec 1 \prec 4 \prec 5$.

2.3 Feasibility

In order to check whether a given set $\mathcal{W}$ of AND/OR precedence constraints is feasible we try to construct a linear realization L in a greedy way: While there exists a job $i \in V$ that is not a waiting job of any of the waiting conditions in $\mathcal{W}$, it is inserted at the end of L. Whenever a waiting condition (X, j) becomes satisfied (which is the case if some $i \in X$ is being added to L), (X, j) is deleted from $\mathcal{W}$. Computational details are provided in Algorithm 1. We use a data structure Q to temporarily store jobs from V. Implementing Q as a stack or a queue leads to a linear time algorithm.

Theorem 2.3.1. *A set of* AND/OR *precedence constraints is feasible if and only if the list L obtained from Algorithm 1 contains all jobs of V.*

Proof. If L contains all jobs, it follows from the construction of Algorithm 1 that, for each waiting condition $(X, j) \in \mathcal{W}$, there is at least one job $i \in X$ with $i \prec_L j$; therefore, according to (*), L is a linear realization. Suppose now that the algorithm returns an incomplete list L although the set of waiting conditions

is feasible. Consider a linear realization $R = (V, E)$ and let $j \in V \setminus L$ be minimal with respect to the total order E. Since the algorithm was not able to add j to L, there is a waiting condition $(X, j) \in \mathcal{W}$ with $X \subseteq V \setminus L$. Since R is a realization, there exists a job $i \in X$ with $(i, j) \in E$ which contradicts the minimal choice of j. □

As a consequence of Theorem 2.3.1 we can formulate the following structural characterization of feasible waiting conditions which appears implicitly already in the work of Igelmund and Radermacher (1983a) within the context of stochastic resource-constrained project scheduling.

Lemma 2.3.2. *A set of* AND/OR *precedence constraints is feasible if and only if there exists no generalized cycle in the associated digraph* D.

Note that Example 2.2.1 is feasible (recall that we already stated two linear realizations). However, if $w_1 = (\{1, 5\}, 4)$ is replaced by $(\{5\}, 4)$ the instance becomes infeasible because $V' = \{4, 5\}$ and $\mathcal{W}' = \{(\{5\}, 4), (\{4\}, 5)\}$ form a generalized cycle.

The following corollary states an algorithmic consequence of Lemma 2.3.2.

Corollary 2.3.3. *Job* $j \in V$ *is not contained in the list* L *returned by Algorithm 1 if and only if* j *is contained in a set* $V' \subseteq V$ *such that for all* $i \in V'$ *there is a waiting condition* $(X, i) \in \mathcal{W}$ *with* $X \subseteq V'$.

In particular, L as a *set* does not depend on the individual jobs chosen from Q in the while-loop of Algorithm 1.

In the proof of Theorem 2.3.1 we have shown that, for a feasible set of AND/OR precedence constraints $\mathcal{W}$, the list L returned by Algorithm 1 is a linear realization of $\mathcal{W}$. In fact, it is an easy observation that Algorithm 1 can generate every linear realization of $\mathcal{W}$ through an appropriate choice of jobs from Q in the while-loop.

Relationship to HORN-SAT. An alternative view of the problem of checking feasibility of $\mathcal{W}$ is to solve a satisfiability problem (SAT) where each clause is of *Horn* type. It can be decided in linear time if such an instance of SAT is satisfiable, see (Dowling and Gallier 1984).

In order to check feasibility, we must decide whether there is a generalized cycle in the digraph D; see Lemma 2.3.2. We assume that there is a dummy job t in V which has to wait for all other jobs in V. Notice that $\mathcal{W}$ is infeasible if and only if t is contained in a generalized cycle. We introduce Boolean variables x_j for every $j \in V$ and y_w for every waiting condition $w \in \mathcal{W}$ with the meaning that $x_j = \mathit{false}$ ($y_w = \mathit{false}$) implies that j (w) is contained in a generalized cycle. Thus, x_j can only be set to true if $y_w = \mathit{true}$ for all $w = (X, j) \in \mathcal{W}$. Contrarily,

$y_w = true$ if there is at least one $i \in X$ with $x_i = true$. These requirements form a set of *implications*

$$\begin{aligned} \bigwedge_{i \in X} (x_i \Rightarrow y_w) \qquad & w = (X, j) \in \mathcal{W}, \\ (\bigwedge_{w=(X,j)} y_w) \Rightarrow x_j \qquad & j \in V, \end{aligned} \tag{2.1}$$

where, if transformed to its conjunctive normal form, each clause has at most one positive literal, and is thus of Horn type. Moreover, if we fix $x_t := false$ and $x_j := true$ for all $j \in V$ with no incoming arcs in D, we obtain the following lemma.

Lemma 2.3.4. *The instance of* HORN-SAT *defined by* (2.1) *with* $x_t := false$ *and* $x_j := true$ *for all* $j \in V$ *with no incoming arcs in* D *is satisfiable if and only if* $\mathcal{W}$ *is infeasible.*

Proof. Let $((x_j)_{j \in V}, (y_w)_{w \in \mathcal{W}})$ be a satisfying assignment; set $V' := \{j \in V \mid x_j = false\}$ and $\mathcal{W}' := \{w \in \mathcal{W} \mid y_w = false\}$. It follows from (2.1) that the subgraph D' of D induced by the set $V' \cup \mathcal{W}'$ of nodes forms a generalized cycle.

On the other hand, if $\mathcal{W}$ is infeasible, denote by $V' \cup \mathcal{W}'$ the set of nodes of D that are contained in some generalized cycle containing t. It is easy to check that

$$x_j := \begin{cases} false & \text{if } j \in V', \\ true & \text{if } j \notin V', \end{cases} \qquad y_j := \begin{cases} false & \text{if } j \in \mathcal{W}', \\ true & \text{if } j \notin \mathcal{W}', \end{cases}$$

is a satisfying assignment. □

2.4 Detecting Implicit AND/OR Precedence Constraints

2.4.1 Problem Definition and Related Work

We now focus on detecting 'new' waiting conditions that can be deduced from the given set $\mathcal{W}$ of constraints. For $U \subset V$ and $j \in V \setminus U$, we say that the waiting condition (U, j) is *implied* by $\mathcal{W}$ if and only if,

for every realization $R = (V, E)$ of $\mathcal{W}$,
there exists some $i \in U$ with $(i, j) \in E$. (**)

By property (*), this is equivalent to the requirement that adding the waiting condition (U, j) to $\mathcal{W}$ does not change the set of realizations for $\mathcal{W}$. Notice that it is sufficient to claim property (**) for every *linear* realization of $\mathcal{W}$.

For traditional precedence constraints the detection of implied waiting conditions is an easy task because the transitive closure which represents all such implicit constraints can be efficiently computed by standard graph algorithms. However, in general, the total number of implicit AND/OR precedence constraints is exponential in the input size of V and $\mathcal{W}$. In particular, it is not possible to compute all implicit constraints efficiently. For the restricted case of AND/OR precedence constraints where the associated digraph D is acyclic, Gillies (1993) proposes an algorithm to determine jobs that have to wait for a single job i.

In the context of directed hypergraphs, Ausiello, d'Atri, and Saccà (1983) (see also Ausiello, d'Atri, and Saccà (1986)) consider problems similar to those discussed in this section and in Section 2.5 below. However, the results we present are not contained in their work because their definition of implicit hyperarcs differs from our definition of implicit waiting conditions. Their definition is based on three rules which are known as 'Armstrong's Axioms' within the context of functional dependencies in relational databases (see, e. g., (Ullman 1982)). In particular, the definition of Ausiello, d'Atri, and Saccà (1983, Definition 4) does not cover implications that can be deduced from the requirement of feasibility. For instance, in Example 2.2.1, the waiting condition $(\{1\}, 4)$ is implied by $\mathcal{W}$ but it is not implied according to the definition of Ausiello, d'Atri, and Saccà (1983).

2.4.2 Result

For a given set $U \subseteq V$ we show that Algorithm 1 can be used to detect all implicit waiting conditions of the form (U, j). For an arbitrary subset $Y \subseteq V$ the set $\mathcal{W}_Y$ of *induced* waiting conditions is given by $\mathcal{W}_Y := \{(X \cap Y, j) \mid (X, j) \in \mathcal{W}, j \in Y\}$. For $(X, j) \in \mathcal{W}$ with $j \in Y$ and $X \cap Y = \emptyset$, the resulting waiting condition $(\emptyset, j) \in \mathcal{W}_Y$ means that job j cannot be planned at all with respect to $\mathcal{W}_Y$; in particular, $\mathcal{W}_Y$ is infeasible in this case.

Theorem 2.4.1. *For given $U \subset V$ let L be the output of Algorithm 1 with input $V \setminus U$ and $\mathcal{W}_{V \setminus U}$. The set of waiting conditions of the form (U, j) which are implied by $\mathcal{W}$ is precisely $\{(U, j) \mid j \in V \setminus (L \cup U)\}$.*

The proof is deferred to Section 2.4.3 below. For Example 2.2.1 and $U := \{2, 3\}$, the algorithm computes $L = \{1\}$ while for $U := \{1, 2\}$ we obtain $L = \{3, 6, 7\}$. Thus, the waiting condition $(\{2, 3\}, 7)$ is implied by $\mathcal{W}$ while $(\{1, 2\}, 7)$ is not.

We can directly deduce the following corollary.

Corollary 2.4.2. *Given $U \subset V$, the set of waiting conditions of the form (U, j) that are implied by $\mathcal{W}$ can be computed in linear time.*

2.4.3 Correctness

We next state some rather technical lemmas which directly show the validity of Theorem 2.4.1. The theorem can alternatively be proved by a simpler argumentation (similar to the proof of Theorem 2.3.1), but we need the lemmas to establish other results in Section 2.5 below. In addition, with the extended argumentation, we are able to slightly strengthen Theorem 2.4.1 (see Corollary 2.4.6). The following definition will be useful throughout the discussion. For a given feasible set $\mathcal{W}$ of waiting conditions and a set $U \subseteq V$ let

$$\begin{aligned} Y_U &:= \{j \in V \setminus U \mid (U,j) \text{ is not implied by } \mathcal{W}\} \qquad \text{and} \\ Z_U &:= \{j \in V \setminus U \mid (U,j) \text{ is implied by } \mathcal{W}\} \;=\; V \setminus (U \cup Y_U) \;. \end{aligned}$$

Lemma 2.4.3. *Let $\mathcal{W}$ be a feasible set of waiting conditions and let $U \subseteq V$. Then, there exists a (linear) realization $R = (V, E)$ of $\mathcal{W}$ such that Y_U is an order ideal of R, i. e., R 'starts' with the jobs in Y_U.*

Proof. Let $R' = (V, E')$ be a linear realization of $\mathcal{W}$ that maximizes the cardinality of the largest order ideal J of R' with $J \subseteq Y_U$. To show that $J = Y_U$, by contradiction, we assume that there is a job $j' \in Y_U \setminus J$. Since, by definition of Y_U, the waiting condition (U, j') is not implied by $\mathcal{W}$, there is a linear realization $R = (V, E)$ of $\mathcal{W}$ with $j' \prec_E U$ (j' precedes all elements in U). Let $j \in Y_U \setminus J$ be minimal with respect to R. By maximality of J, job j cannot be moved to the position directly after J in R' without violating a waiting condition. Thus, there exists $(X, j) \in \mathcal{W}$ with $X \subseteq V \setminus J$. By (*), there exists an $i \in X$ with $(i, j) \in E$. Notice that we have $i \notin U$ because $i \prec_E j \preceq_E j' \prec_E U$. Moreover, due to the minimal choice of j and the fact that $i \notin J$ it follows that $i \notin Y_U$. As a consequence we obtain $i \in X \setminus (U \cup Y_U)$. By definition of Z_U, this yields $i \in Z_U$. Thus, the waiting condition (U, i) is implied by $\mathcal{W}$ which is a contradiction to $i \prec_E j \preceq_E j' \prec_E U$. □

Let us call a set $Y \subseteq V$ *feasible with respect to $\mathcal{W}$* if and only if the induced set $\mathcal{W}_Y$ of waiting conditions is feasible. The result in Corollary 2.3.3 can then be restated as follows.

Corollary 2.4.4. *Algorithm 1 returns the unique maximal feasible subset of V with respect to $\mathcal{W}$.*

In conjunction with the following lemma, Corollary 2.4.4 provides an efficient way of detecting waiting conditions implied by $\mathcal{W}$. This concludes the proof of Theorem 2.4.1 (in fact, the lemma essentially is a reformulation of Theorem 2.4.1).

Lemma 2.4.5. *Let $\mathcal{W}$ be a feasible set of* AND/OR *precedence constraints, $U \subset V$, and $j \in V \setminus U$. Then the waiting condition (U, j) is implied by $\mathcal{W}$ if and only if j is not contained in the unique maximal feasible subset of $V \setminus U$ with respect to $\mathcal{W}_{V \setminus U}$.*

Proof. We have to show that Y_U is the unique maximal feasible subset F of $V \setminus U$ with respect to $\mathcal{W}_{V \setminus U}$. By Lemma 2.4.3, there exists a linear realization R of $\mathcal{W}$ starting with the jobs in Y_U. This induces a linear realization of $\mathcal{W}_{Y_U}$ and consequently, by definition, Y_U is feasible with respect to $\mathcal{W}$. Moreover, since $\mathcal{W}_{Y_U} = (\mathcal{W}_{V \setminus U})_{Y_U}$, Y_U is feasible with respect to $\mathcal{W}_{V \setminus U}$ which yields $Y_U \subseteq F$.

To show that $F \subseteq Y_U$, by contradiction, assume that $F \setminus Y_U \neq \emptyset$. Since $F \cap U = \emptyset$ we then have $F \cap Z_U \neq \emptyset$. Let $R' = (F, E')$ be a linear realization of $\mathcal{W}_F$ and choose $i \in F \cap Z_U$ minimal with respect to R'. Since $i \in Z_U$, by definition of Z_U, (U, i) is implied by $\mathcal{W}$. Therefore, moving job i to the position directly after Y_U in R violates a waiting condition $(X, i) \in \mathcal{W}$ with $X \subseteq U \cup Z_U$. Let us consider the induced waiting condition $(X \cap F, i) \in \mathcal{W}_F$. It follows from the minimal choice of i that $i' \notin F \cap Z_U$ for all i' with $(i', i) \in E'$. With $F \cap U = \emptyset$, this implies $i' \notin X \cap F$ which is a contradiction to the fact that R' is a realization for $\mathcal{W}_F$. □

Finally, notice that there may exist (implicit) waiting conditions (X, j) inside the considered set U, i. e., $X \subset U$ and $j \in U \setminus X$. Theorem 2.4.1 can be strengthened in the following way. Consider the situation after the execution of Algorithm 1 with input $V \setminus U$ and $\mathcal{W}_{V \setminus U}$ and let L be the resulting list of jobs. Furthermore, let $U' \subseteq U$ denote the set of jobs from U that can be added to L without violating any waiting condition of $\mathcal{W}$.

Corollary 2.4.6. *For given $U \subseteq V$ the set of waiting conditions (U', j) which is implied by $\mathcal{W}$ is precisely $\{(U', j) \mid j \in V \setminus (L \cup U')\}$.*

Proof. We show that the maximal feasible subsets F and F' of $V \setminus U$ and $V \setminus U'$, respectively, coincide. The corollary then follows from Lemma 2.4.5. It is clear that $F \subseteq F'$ since $U' \subseteq U$. Conversely, suppose by contradiction that $F' \setminus F \neq \emptyset$. Denote by $R = (V, E)$ and $R' = (V, E')$ linear realizations of $\mathcal{W}$ where F and F' are order ideals, respectively (the existence of R and R' follows from Lemma 2.4.3 and 2.4.5). Let $j \in F' \setminus F$ be the smallest job in $F' \setminus F$ with respect to R'. Since $j \notin F$, moving job j in R to the position directly after F violates a waiting condition (X, j) with $X \cap F = \emptyset$. Since R' is a realization there must exist some $i \in X$ with $(i, j) \in E'$. But $i \in F' \setminus F$ which contradicts the minimal choice of j. □

2.5 Minimal Representation of AND/OR Precedence Constraints

While for traditional precedence constraints a minimal representation without redundancies is given by the transitive reduction and can be computed by simply removing redundant (i. e., transitive) constraints, the situation is slightly more complicated for AND/OR precedence constraints. In order to obtain a *unique* minimal representation, it is not sufficient to iteratively remove redundant waiting conditions that are implied by the others.

Definition 2.5.1. *A set $\mathcal{W}$ of waiting conditions is called* minimal *if*

i) no waiting condition $(X, j) \in \mathcal{W}$ is implied by $\mathcal{W} \setminus \{(X, j)\}$ and

ii) for each waiting condition $(X, j) \in \mathcal{W}$, the set X is minimal with respect to inclusion, i. e., for all $i \in X$, the waiting condition $(X \setminus \{i\}, j)$ is not implied by $\mathcal{W}$.

Two sets $\mathcal{W}$ and $\mathcal{W}'$ of waiting conditions are called equivalent *if their sets of (linear) realizations coincide. Moreover, if $\mathcal{W}'$ is minimal, then $\mathcal{W}'$ is called a* minimal reduction *of $\mathcal{W}$.*

The set $\mathcal{W}$ from Example 2.2.1 is not minimal: If $(\{4, 5, 6\}, 7)$ is replaced by $(\{4, 6\}, 7)$ the resulting instance is equivalent to Example 2.2.1. This follows from waiting condition $(\{4\}, 5)$ which ensures that whenever $(5, 7) \in E$ in some realization $R = (V, E)$ we also have $(4, 5) \in E$ and $(4, 7) \in E$. Note that if we additionally replace $(\{1, 5\}, 4)$ by $(\{1\}, 4)$ the resulting set of waiting conditions is minimal (and still equivalent to Example 2.2.1).

Theorem 2.5.2. *Each feasible set of waiting conditions has a unique minimal reduction.*

To prove the theorem we need the following technical lemma.

Lemma 2.5.3. *Let $\mathcal{W}$ be a feasible set of waiting conditions with $(U, j) \in \mathcal{W}$. (U, j) is implied by $\mathcal{W}' := \mathcal{W} \setminus \{(U, j)\}$ if and only if there exists some $(X, j) \in \mathcal{W}'$ with $X \subseteq U \cup Z_U$.*

Proof. If (U, j) is implied by $\mathcal{W}'$ then any ordering of V where Y_U is an ideal and j is placed directly after Y_U is not a realization of $\mathcal{W}'$. Thus, there exists some $(X, j) \in \mathcal{W}$ with $X \subseteq U \cup Z_U$. Contrarily, suppose that there exists some $(X, j) \in \mathcal{W}'$ with $X \subseteq U \cup Z_U$ but (U, j) is not implied by $\mathcal{W}'$. Then there exists some $h \in X \setminus U$ and a linear realization $R' = (V, E')$ of $\mathcal{W}'$ with $Y_U \prec_{E'} h \prec_{E'} j \prec_{E'} U$. Since $\mathcal{W}$ is feasible, there exists some $i \in U$ that can be moved to the position directly after h without violating a waiting condition of

$\mathcal{W}'$ (this follows from Corollary 2.4.6). In addition, the resulting linear realization $R = (V, E)$ satisfies the waiting condition (U, j) and thus should be a realization with respect to $\mathcal{W}$. However, we have $h \prec_E i \prec_E j \prec_E U \setminus \{i\}$ which is a contradiction to $h \in Z_U$. □

Proof of Theorem 2.5.2. Let $\mathcal{W}$ and $\mathcal{W}'$ be equivalent and both minimal. It suffices to show that $(U, j) \in \mathcal{W}$ implies $(U, j) \in \mathcal{W}'$. By Lemma 2.4.3, there exists a linear realization R of $\mathcal{W}$ starting with Y_U. Since the order obtained by moving j to the position directly after Y_U in R is not a realization, there exists a waiting condition $(X, j) \in \mathcal{W}'$ with $X \subseteq U \cup Z_U$. We show next that $U \subseteq X$. By minimality of $\mathcal{W}'$ this implies $X = U$ which concludes the proof.

Assume that $X \not\supseteq U$ and let $i \in U \setminus X$. We obtain a linear realization R' by moving i to the position directly after Y_U in R; otherwise, there exists a waiting condition $(Z, i) \in \mathcal{W}$ with $Z \subseteq U \cup Z_U$. Since all jobs in Z_U have to wait for a job in U, the waiting condition $(U \setminus \{i\}, i)$ and thus $(U \setminus \{i\}, j)$ is implied by $\mathcal{W}$ which is a contradiction to the minimality of $\mathcal{W}$.

Since moving j to the position directly after $Y_U \cup \{i\}$ in R' violates the waiting condition (X, j), there exists a waiting condition $(Z, j) \in \mathcal{W}$ with $Z \subseteq (U \setminus \{i\}) \cup Z_U$. However, by Lemma 2.5.3, the set $\mathcal{W} \setminus \{(U, j)\}$ implies the waiting condition (U, j) which is a contradiction to the minimality of $\mathcal{W}$. □

Let us next consider the following straightforward polynomial time algorithm to compute a minimal reduction of a set $\mathcal{W}$ of waiting conditions. For each $(X, j) \in \mathcal{W}$, apply Algorithm 1 with input $V \setminus X$ and $\mathcal{W}_{V \setminus X}$. If besides (X, j) some other waiting condition prevents that j can be added to L, then remove (X, j) from $\mathcal{W}$. Otherwise, remove all i from X which cannot be added to L because some waiting condition of $\mathcal{W}$ is violated. Finally, output the resulting set of waiting conditions. An implementation of this rough scheme is given in Algorithm 2. There, $a(j)$, $j \in V$, denotes the number of waiting conditions of the form (X, j) that are left in $\mathcal{W}_{V \setminus U}$ after Algorithm 1 was called with input $V \setminus U$ and $\mathcal{W}_{V \setminus U}$. Notice that $a(j)$ is computed within the execution of Algorithm 1. In the following theorem we prove the correctness of the algorithm (as defined earlier, A is the set of arcs in the digraph induced by $\mathcal{W}$).

Theorem 2.5.4. *Algorithm 2 computes the minimal reduction of a set $\mathcal{W}$ of waiting conditions in* $O(|\mathcal{W}| \cdot |A|)$ *time.*

Proof. We first show that, through the procedure, the transformed set of waiting conditions is equivalent to $\mathcal{W}$ given as input. We then argue that, once the algorithm has finished, the obtained set of waiting conditions is minimal.

Denote by $\mathcal{W}^k$, $k \in \{1, \ldots, |\mathcal{W}|\}$, the set of waiting conditions after the k-th iteration of the outer for-loop of Algorithm 2. Furthermore, let $\mathcal{W}^0 := \mathcal{W}$.

Algorithm 2: Computation of a minimal reduction

Input : A set V of jobs and waiting conditions $\mathcal{W}$.
Output: A minimal reduction of $\mathcal{W}$

for *each* $(U, j) \in \mathcal{W}$ **do**
  $L :=$ call Algorithm 1 with input $V \setminus U$ and $\mathcal{W}_{V \setminus U}$ and compute $a(i)$ for each $i \in V$;
  if $a(j) > 1$ **then**
    delete (U, j) from $\mathcal{W}$;
  else for $i \in U$ **do**
    if $a(i) > 0$ **then** delete i from U;
return $\mathcal{W}$;

Suppose that some (U, j) is removed from $\mathcal{W}^{k-1}$ in the k-th iteration of the algorithm. Since $a(j) > 1$ in the k-th iteration there exists a waiting condition $(X, j) \in \mathcal{W}^{k-1}$ with $X \neq U$ and $X \subset U \cup Z_U$. With Lemma 2.5.3, (U, j) is implied by $\mathcal{W}^k = \mathcal{W}^{k-1} \setminus \{(U, j)\}$ and can thus be deleted from $\mathcal{W}^{k-1}$. Now assume that, in (U, j), some job i was deleted from U in the k-th iteration. Then $a(i) > 0$ and with Corollary 2.4.6 it follows that $(U \setminus \{i\}, i)$ is implied by $\mathcal{W}^{k-1}$. Together with (U, j) this shows that $(U \setminus \{i\}, j)$ is implied by $\mathcal{W}^{k-1}$. Thus, $\mathcal{W}^{k-1}$ is equivalent to $\mathcal{W}^k$ for all $k \in \{1, \ldots, |\mathcal{W}|\}$ which directly implies that $\mathcal{W}$ and $\mathcal{W}' := \mathcal{W}^{|\mathcal{W}|}$ are equivalent.

We now show that $\mathcal{W}'$ is minimal. Let us first suppose that some $(U, j) \in \mathcal{W}'$ is implied by $\mathcal{W}' \setminus \{(U, j)\}$. Then, by Lemma 2.5.3, there exists another waiting condition $(X, j) \in \mathcal{W}' \setminus \{(U, j)\}$ with $X \subseteq U \cup Z_U$. Notice that Z_U in dependence of $\mathcal{W}' \setminus \{(U, j)\}$ and all $\mathcal{W}^k$, $k \in \{0, \ldots, |\mathcal{W}|\}$ is constant because the associated sets of realizations coincide. The waiting conditions (U, j) and (X, j) have been constructed in some iterations k and k', respectively, in which waiting conditions $(U', j) \in \mathcal{W}$ with $U \subseteq U'$ and $(X', j) \in \mathcal{W}$ with $X \subseteq X'$ have been treated by the algorithm. If $(X, j) \in \mathcal{W}^{k-1}$ then, by Lemma 2.5.3, (U', j) would have been removed from $\mathcal{W}^{k-1}$. Consequently, $k' > k$. Since U was obtained from U' (in the k-th iteration of the algorithm), by Corollary 2.4.6, there exists a linear realization R which starts with Y_U followed by first an *arbitrary* job $i \in U$ and then job j. Since, by assumption, $\mathcal{W}^{k-1}$ and $\mathcal{W}' \setminus \{(U, j)\}$ are equivalent, (X, j) must be respected by R; hence $U \subseteq X$. But then (X', j) is deleted in iteration $k' > k$, a contradiction. Next, suppose that $\mathcal{W}'$ contains a waiting condition (U, j) such that, for some $i \in U$, the waiting condition $(U \setminus \{i\}, j)$ is implied by $\mathcal{W}'$. Since i was not removed from U' in the k-th iteration of the algorithm, it follows from Corollary 2.4.6 that $(U' \setminus \{i\}, j)$ is not implied by $\mathcal{W}^{k-1}$. Thus

there exists a linear realization $R = (V, E)$ of $\mathcal{W}^{k-1}$ with $j \prec_E (U' \setminus \{i\})$ and in particular $j \prec_E (U \setminus \{i\})$. Since R is also a realization for $\mathcal{W}'$ the waiting condition $(U \setminus \{i\}, j)$ is not implied by $\mathcal{W}'$ — a contradiction.

The above argumentation shows that $\mathcal{W}'$ is minimal and thus, Algorithm 2 computes a minimal reduction of $\mathcal{W}$. Finally, the running time follows from the fact that Algorithm 1 is called $|\mathcal{W}|$ times. □

Notice that the cardinality of a minimal set of waiting conditions might still be exponential in the number of jobs $|V|$: Let $V = \{1, 2, \ldots, 2\ell + 1\}$; in order to model the constraint that job $2\ell + 1$ can only be planned after at least ℓ other jobs, we need exactly $\binom{2\ell}{\ell}$ waiting conditions.

2.6 An NP-Complete Generalization

Suppose that we generalize the definition of waiting conditions from (X, j), $X \subset V$, $j \in V \setminus X$ to (X, X') with $X, X' \subset V$ and $X \cap X' = \emptyset$. The generalized waiting condition (X, X') is fulfilled if at least one job $j \in X'$ is waiting for at least one job $i \in X$. We show in the theorem below that the problems considered in Sections 2.3 and 2.4 become NP-complete for this generalized setting.

Theorem 2.6.1. *Given a set of jobs with generalized waiting conditions, it is already NP-complete to decide whether or not a waiting condition $(\{i\}, \{j\})$ is implied for two jobs i and j.*

Proof. We construct a reduction from the satisfiability problem SAT. The construction is depicted in Figure 2.2. Given an instance of SAT, we introduce for each Boolean variable x two jobs which correspond to the two literals x and $\bar{x}$ (negation of x); to keep notation simple, we denote these jobs also by x and $\bar{x}$. Moreover, for each clause C we introduce a corresponding job (also denoted by C) and a waiting condition $(X_C, \{C\})$ where X_C denotes the set of literals in clause C; in other words, job C may not be started before at least one job corresponding to a literal of clause C has been completed. Finally, we introduce two additional jobs a and b together with the following waiting conditions: For each variable x, at least one of the jobs x and $\bar{x}$ has to wait for a, i. e., we have the waiting condition $(\{a\}, \{x, \bar{x}\})$. For each clause C, b has to wait for the corresponding job, which is given by the waiting condition $(\{C\}, \{b\})$.

It is easy to check that in the constructed scheduling instance job b has to wait for job a if and only if the underlying instance of SAT does not have a satisfying truth assignment. If there is a satisfying truth assignment, then we can construct a linear realization where b precedes a in the following way: First we take all jobs corresponding to literals with value 'true' in an arbitrary order, next we append

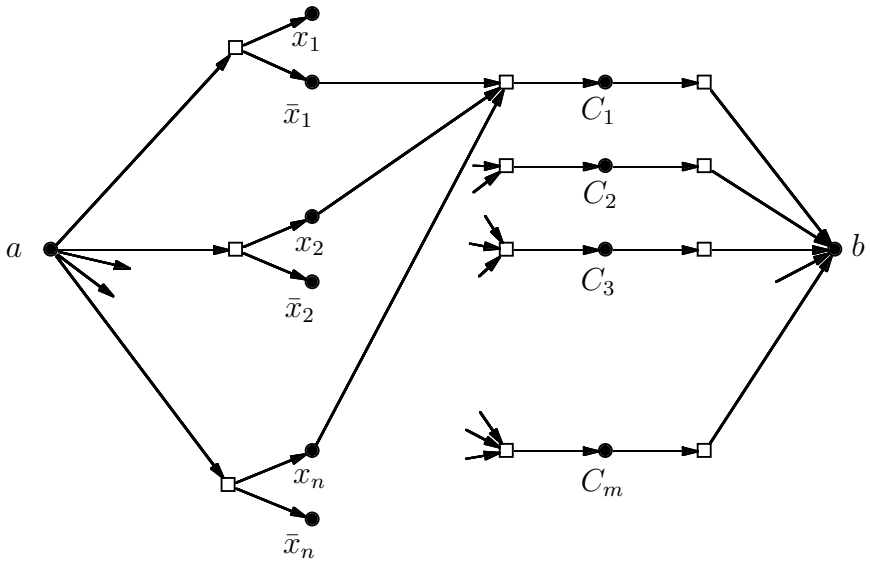

Figure 2.2: The figure shows the construction from an instance of SAT to a set of generalized waiting conditions as described in the proof of Theorem 2.6.1. In the figure, as usual, we have a node for each job and for each (generalized) waiting condition. In extension to the graphical representation of waiting conditions (X, j) we visualize each set X' of a generalized waiting condition $w = (X, X')$ by $|X'|$ outgoing arcs (w, j), $j \in X'$, of w. As an example, according to the figure, the clause C_1 is $\bar{x}_1 \vee x_2 \vee x_n$.

all jobs corresponding to clauses in some order, afterwards we add b, then a, and finally all remaining jobs corresponding to literals with the value 'false'. On the other hand, if there is a linear realization where b precedes a, we can define a corresponding satisfying truth assignment in the following way: For each variable x, assign x the value true (false) if the job corresponding to x ($\bar{x}$) precedes a; notice that at most one of the two cases can happen, if neither job x nor $\bar{x}$ precedes a, we assign an arbitrary value to the variable x. □

As a consequence of Theorem 2.6.1, we obtain that the problem of deciding feasibility is also NP-complete for this generalized model.

CHAPTER 3

AND/OR PRECEDENCE CONSTRAINTS: COMPUTING EARLIEST JOB START TIMES

The chapter is concerned with the computation of earliest start times of jobs that have to be scheduled with respect to AND/OR precedence constraints. Like the results of the previous chapter, some part of the developed theory will later turn out to be a fundamental component of our approach to solve stochastic resource-constrained project scheduling problems.

We consider different ranges of minimal *time distances* (or time lags) between the start times of two jobs i and j that are coupled within an AND/OR precedence constraint. For the cases that such time lags are strictly positive or non-negative, we propose polynomial time algorithms of different time complexity. Generally, the problem is to find a solution to a system of *min-max-inequalities* which has a number of applications in other fields of research. As an example, we discuss a strong relationship to finding optimal strategies for a class of 2-person games played on directed graphs.

Finally, we study the time-cost tradeoff problem for the case of AND/OR precedence constraints. We discuss the analytical properties of the time-cost tradeoff curve and show the NP-hardness of computing a single point of the curve. Except for the study of the time-cost tradeoff problem, the chapter is based on the second part of the paper (Möhring, Skutella, and Stork 2000b).

3.1 Problem Definition and Related Work

For this chapter we assume that together with each waiting condition $w = (X, j) \in \mathcal{W}$ and each job $i \in X$ we are given an integral time lag $-M < d_{iw} < M$, $M \geqslant 0$. We aim at finding a vector of earliest start times $S = (S_1, \ldots, S_n)$ such that for each waiting condition $(X, j) \in \mathcal{W}$ the constraint

$$S_j \;\geqslant\; \min_{i \in X}(S_i + d_{iw}) \tag{3.1}$$

is satisfied. Notice that job processing times p_i can be modeled by setting $d_{iw} := p_i$ for all $w = (X, j)$ and $i \in X$. Negative values d_{iw} represent so-called *maximal*

time lags that define latest possible start times of jobs $i \in X$ relative to j. Such negative time lags allow to model, e. g., job overlappings or job synchronizations.

In order to simplify the presentation, we sometimes interpret nodes of the digraph D that represent waiting conditions as dummy jobs. We then assume that the vector S also contains start times of these dummy jobs and constraint (3.1) is replaced by

$$S_w \;\geqslant\; \min_{i \in X}(S_i + d_{iw}) \qquad \text{and} \qquad S_j \;\geqslant\; S_w \;\;.$$

We call the jobs in V AND-*nodes* and the jobs in $\mathcal{W}$ OR-*nodes* of the digraph D. We assume that a dummy AND-node a precedes all other AND-nodes, i. e., we introduce a waiting condition $(\{a\}, j)$ for all $j \in V$. In D, time lags can easily be integrated by associating each d_{iw} as a weight to the arc (i, w), see Figure 2.1. Moreover, we always associate the weight $d_{wj} = 0$ to all arcs (w, j), $w = (X, j) \in \mathcal{W}$, i. e., the time lag between a dummy job w and its successor j is 0. The case of $d_{wj} \neq 0$ can be handled by replacing $d_{wj} \neq 0$ by 0 and d_{iw} by $d_{iw} + d_{wj}$ for all $i \in in(w)$.

The problem of finding earliest start times can then be formulated on D as follows: Find a component-wise minimal schedule $S \in \mathbb{Z}^{|\mathcal{V}|}$ fulfilling $S_a \geqslant 0$ and

$$\begin{aligned} S_j &\geqslant \max_{(w,j) \in A}(S_w + d_{wj}) \qquad j \in V, \\ S_w &\geqslant \min_{(j,w) \in A}(S_j + d_{jw}) \qquad w \in \mathcal{W}\,. \end{aligned} \tag{ES}$$

As we noted above, in the sequel we assume that $d_{wj} = 0$. Besides schedules $S \in \mathbb{Z}^{|\mathcal{V}|}$ we also consider *partial schedules* $S \in (\mathbb{Z} \cup \{\infty\})^{|\mathcal{V}|}$ where the start time of a job may be infinite with the meaning that the job is not planned. As usual, a (partial) schedule fulfilling the constraints of (ES) is called *feasible*. In particular, the partial schedule $S = (\infty, \ldots, \infty)$ fulfills all inequalities of (ES) and is thus feasible. Moreover, it is easy to see that, if S' and S'' are feasible partial schedules, then their component-wise minimum $S := \min\{S', S''\}$ is also feasible. In particular, there always exists a (unique) component-wise minimal partial schedule S^*, called the *optimal* partial schedule (notice that $S^* \geqslant 0$ for all AND-nodes). It follows that, instead of considering the above system of inequalities, we alternatively may consider the corresponding system of equations (which is obtained from (ES) by replacing each '$\geqslant$' by a '$=$').

Presuming different restrictions on the range of arc weights, several algorithms have been suggested to solve (ES). Note that all restrictions on arc weights are meant to refer to arcs (j, w) between AND-nodes j and OR-nodes w, only. For the case of positive arc weights, a modification of Dijkstra's shortest path algorithm can be applied. An algorithm suggested by Knuth (1977) has running time

O($|\mathcal{V}| \log |\mathcal{V}| + |A|$). Other approaches are proposed by Dinic (1990), Gallo, Longo, Pallottino, and Nguyen (1993), and, in the context of resource-constrained project scheduling, by Igelmund and Radermacher (1983a) (see also Chapter 5 below). Levner, Sung, and Vlach (1999) consider a generalized model of AND/OR precedence constraints where a so-called *threshold value* $1 \leqslant \ell_w \leqslant |X|$ is associated with each waiting condition $w = (X, j)$, indicating that j may start if at least ℓ_w jobs from X have been completed. They show that Dijkstra's shortest path algorithm can also be generalized to solve their model (with positive arc weights). The general case $-M < d_{jw} < M$ is a frequently studied problem with applications in many different areas, e. g., *game theory* (Zwick and Paterson 1996) and *interface timing verification* (see (Schwiegelshohn and Thiele 1999) and (McMillan and Dill 1992)). Moreover, there are applications stemming from online optimization, see (Zwick and Paterson 1996, Section 7) for a collection of examples. Interestingly, although a pseudo-polynomial algorithm to solve this case of (ES) is easily obtained, no algorithm polynomial in $|\mathcal{V}|$ and $\log(M)$ is currently known.

3.2 Arbitrary Arc Weights

In this section we study the case of arbitrary arc weights $-M < d_{jw} < M$.

3.2.1 Feasibility

For arbitrary arc weights the feasibility results stated in Section 2.3 are not valid anymore. They are based on the requirement that all $d_{jw} = p_j > 0$, and consequently, an AND-node j can start if and only if for all $(X, j) \in \mathcal{W}$ at least one $i \in X$ has previously been started (compare to condition (*) in Section 2.2). However, if we allow $d_{jw} \leqslant 0$, this is no longer the case. We demonstrate this on the well-studied model of AND-constraints with arbitrary time lags. Consider two jobs with AND-constraints $w_1 = (\{1\}, 2)$ and $w_2 = (\{2\}, 1)$ with $d_{1w_1} \geqslant 0$ and $d_{1w_1} + d_{2w_2} \leqslant 0$. According to Lemma 2.3.2 the instance is infeasible, however, $S_1 = 0$ and $S_2 = d_{1w_1}$ obviously is a feasible schedule. In the sequel we derive a necessary and sufficient feasibility criterion for (ES) with arbitrary arc weights which generalizes the feasibility criterion given in Lemma 2.3.2. For the remainder of the chapter we call a set $\mathcal{W}$ of waiting conditions *feasible* if and only if there exists a feasible schedule for (ES).

Before we can derive the criterion, we discuss how a given instance can be simplified without changing the optimal partial schedule S^*. First, we make the problem more restrictive by removing all but one incoming arc of each OR-node w. If the remaining arc (j, w) fulfills $S^*_j + d_{jw} \leqslant S^*_w$, clearly, all inequalities of

(ES) are still satisfied. Consequently, S^* and the optimal (partial) schedule of the more restrictive instance coincide. In a similar fashion we can remove all but one incoming arc (w, j) of each AND-node j without changing the earliest start times. However, removing such arcs means relaxing the problem and some more work has to be done in order to obtain the wanted result.

Lemma 3.2.1. *For each digraph D representing a set of* AND/OR *precedence constraints, there exists a sub-digraph $\bar{D}$ on the same set $\mathcal{V}$ of nodes with $|in_{\bar{D}}(j)| \leqslant 1$ for all* AND*-nodes $j \in V$ such that $S^* = \bar{S}^*$, where $\bar{S}^*$ denotes the optimal (partial) schedule of $\bar{D}$.*

Proof. We construct $\bar{D}$ by iteratively removing arcs (w, j) from D that do not affect the earliest start time of the AND-node j. By contradiction, assume that, once all such arcs have been removed, there is some AND-node j with $|in_{\bar{D}}(j)| > 1$. Thus, removing any incoming arc of j reduces the earliest start time of j. Denote by S^1 and S^2 the optimal (partial) schedules obtained if two different incoming arcs (w_1, j) and (w_2, j) are removed from $\bar{D}$; then, $S^*_j > S^1_j$ and $S^*_j > S^2_j$. Without loss of generality let $S^1_j \leqslant S^2_j$; we define a new (partial) schedule S through

$$S_i \;:=\; \min\{S^1_i + S^2_j - S^1_j, S^2_i\} \qquad \text{for all } i \in \mathcal{V}.$$

By definition, $S \leqslant S^2$ and $S_j = S^2_j$. For each arc (w, j) with $w \neq w_2$ we have $S^2_j \geqslant S^2_w$ which yields $S_j = S^2_j \geqslant S^2_w \geqslant S_w$. For $w = w_2$ we get $S_j = S^1_j + S^2_j - S^1_j \geqslant S^1_w + S^2_j - S^1_j \geqslant S_w$. We obtain $S_j \geqslant \max_{(w,j)\in A}(S_w)$. Furthermore, S also fulfills all other inequalities of (ES), because both $S^1 + S^2_j - S^1_j$ and S^2 fulfill the inequalities and so does its minimum S. Consequently, S is a feasible (partial) schedule which is a contradiction to the minimality of S^* since $S_j < S^*_j$. □

For the subsequent presentation, recall the definition of a (generalized) cycle from Section 2.2. Note that we assume all cycles to be directed cycles.

Corollary 3.2.2. *Let $\bar{D}$ be as in Lemma 3.2.1. Then, all cycles in $\bar{D}$ have strictly positive length.*

Proof. Assume that there is a cycle $(w_1, j_1, w_2, j_2, \ldots, w_k, j_k, w_1)$ of non-positive length in $\bar{D}$. By definition of $\bar{D}$, (w_ℓ, j_ℓ) is the only incoming arc for node j_ℓ, $\ell = 1, \ldots, k$. Thus, one can construct a feasible partial schedule for $\bar{D}$ satisfying

$$S_{w_1} = S_{j_1} = -1 \qquad \text{and} \qquad S_{w_\ell} = S_{j_\ell} = -1 + \sum_{q=2}^{\ell} d_{j_{q-1}w_q} \quad \text{for } \ell = 2, \ldots, k.$$

With Lemma 3.2.1, this is also possible for the original digraph D which yields a contradiction to the requirement $S^*_{j_1} \geqslant 0$. □

Lemma 3.2.3. *A set of* AND/OR *precedence constraints with arbitrary arc weights is feasible if and only if each generalized cycle in D contains a cycle of non-positive length.*

Proof. Let C be a generalized cycle which contains only cycles of positive length and suppose that some node $v \in C$ can be scheduled at $S_v < \infty$. Since v has at least one incoming arc there must exist a node $u \in in(v)$ with $S_v \geqslant S_u + d_{uv}$ (notice that this is also the case for $S_v = 0$). Iterating this argument, since $|C|$ is finite, we obtain a cycle in C with non-positive length — a contradiction.

Conversely, suppose that the given instance is infeasible. Let $Z \neq \emptyset$ denote the set of nodes whose earliest start times are ∞. By Lemma 3.2.1, we can relax the problem by removing all but one incoming arc of each AND-node such that the earliest start times remain unchanged for the resulting digraph $\bar{D}$. Remove all OR-nodes from Z whose out-degree is 0 in $\bar{D}$ and denote the resulting set of nodes by Z'. Then Z' induces a generalized cycle C in D. Moreover, by definition of Z', every cycle in C is also contained in $\bar{D}$ and has therefore positive length by Corollary 3.2.2. □

Lemma 3.2.3 reduces to Lemma 2.3.2 if all arc weights d_{jw} are strictly positive. Lemma 3.2.3 enables us to show that the decision problem of (ES) is in both NP and co-NP. The decision problem corresponding to (ES) is to decide whether or not a feasible schedule $S < \infty$ for (ES) exists.

Lemma 3.2.4. *The decision problem corresponding to (ES) is in NP ∩ co-NP.*

Proof. It is clear that the decision problem corresponding to (ES) is in NP because, for a given feasible schedule S of some instance I, it is easy to verify all constraints of $\mathcal{W}$. Moreover, it follows from Lemma 3.2.3 that the decision problem corresponding to (ES) is in co-NP. We can guess a generalized cycle violating the condition in Lemma 3.2.3, which can be verified in polynomial time by, e. g., some standard minimum mean weight cycle algorithm. □

3.2.2 A Simple Pseudo-Polynomial Time Algorithm

For the case of (ES) with arbitrary arc weights several pseudo-polynomial algorithms (partly independent of each other) have been proposed, see, e. g., (Chauvet and Proth 1999) and (Schwindt 1998) as well as (Schwiegelshohn and Thiele 1999), and (Zwick and Paterson 1996). A very simple (pseudo-polynomial) algorithm is as follows. First, initialize $S_j := 0$ for all $j \in V$. Then, while S violates some waiting condition $w = (X, j) \in \mathcal{W}$, set $S_j := \min_{i \in X}(S_i + d_{iw})$. If S_j becomes larger than a given time horizon T, then stop and return that the given instance is infeasible. The time horizon T can be chosen as $T :=$

$\sum_{j \in V} (\max_{w \in out(j)} |d_{jw}|)$. One can show straightforwardly by induction that $S \leqslant S^*$ in each iteration of the algorithm. If the recurrence is exited in proper form, all constraints are obviously fulfilled ($S \geqslant S^*$) and thus $S = S^*$. Moreover, at least one start time of a job is increased by 1 in each iteration. Thus the number of iterations is O($|V| \cdot T$). Finding a violated waiting condition obviously requires at most O($|A|$) time and thus the total complexity is O($|V| \cdot |A| \cdot T$). Note that for the special case that D is acyclic, earliest job start times can easily be computed in linear time along a topological sort. Moreover, if each AND-node (OR-node) has at most one incoming arc, node start times can be computed by, e. g., the Bellman-Ford shortest (longest) path algorithm in time O($|V| \cdot |A|$).

3.2.3 A Game-Theoretic Application

We next consider a class of 2-player games played on bipartite directed graphs which are directly related to the problem (ES). There exists substantial literature on different variations of this game, see, e. g., (Zwick and Paterson 1996; Ehrenfeucht and Mycielski 1979; Jurdziński 1998; Vöge and Jurdziński 2000), as well as references therein. Each player is identified with one of the node partitions of the graph. The game starts at a fixed node j_0 and the player associated with that node chooses an arc (w_0, j_0). Then the other player chooses an arc (j_1, w_0) and so on. The objective and the stopping criterion depends on the considered variation of the game. One variation is the so-called *mean payoff game* (MPG) where an integer weight is associated to each arc of the digraph. Furthermore, it is assumed that each node has at least one incoming arc. The mean payoff game is finished as soon as the path P resulting from the game contains a cycle and the *outcome* ν of the game is the mean weight of the arcs of that cycle. One player wants to maximize the outcome while the other player wants to minimize it. It has been shown by Ehrenfeucht and Mycielski (1979) that both players have positional optimal strategies, that is, the decisions of both players do neither depend on previous choices nor on the start node j_0. In the following we always assume that j_0 is associated with the maximization player.

The decision problem corresponding to (MPG) is to decide whether the outcome of the game is positive. Zwick and Paterson (1996) have noted that this problem is in NP $\cap$ co-NP. Even more, Jurdziński (1998) showed that the problem is in UP $\cap$ co-UP. It seems to be intuitively clear that the problems (MPG) and (ES) are closely related. We next show that this is indeed the case.

Lemma 3.2.5. *The decision problems corresponding to (MPG) and (ES) are polynomially equivalent.*

Proof. Given an instance of (ES), we construct an instance of (MPG) in the following way. First, we add an additional job b and a waiting condition $w_j =$

$(\{j\}, b)$ with $d_{jw_j} = 0$ for every job $j \in V$; moreover, we add a waiting condition $w = (\{b\}, a)$ with $d_{bw} = -T$, where T is the time-horizon discussed in Section 3.2.2. Notice that there exists a feasible schedule for the original instance of (ES) if and only if the earliest start time of the new job b is finite. The game digraph D is now the digraph representing the new scheduling instance. The starting node is $j_0 := b$ and the maximization player starts. We show that the set of AND/OR precedence constraints is feasible if and only if $\nu \leqslant 0$.

Only if: Based on an optimal schedule $S^* < \infty$, we give a strategy for the minimization player which ensures $\nu \leqslant 0$: In each OR-node w, choose an incoming arc (j, w) with $S^*_j + d_{jw} = S^*_w$. Then, for two vertices v_1 and v_2 on the path formed by the game, the weight of the (directed) sub-path from v_1 to v_2 is at most $S^*_{v_2} - S^*_{v_1}$ (for each arc (w, j) on the path we have $S^*_j \geqslant S^*_w$ and for each arc (j, w) on the path we have $S^*_w = S^*_j + d_{jw}$). In particular, the length of the cycle terminating the game is at most 0 (choose $v_1 = v_2$).

If: For an infeasible scheduling instance it follows from Lemma 3.2.3 that there exists a generalized cycle C in D which only contains cycles of positive length. Without loss of generality, C contains the node b. We give a strategy for the maximization player which ensures $\nu > 0$: In each step, choose an arc which starts at a node in C. Such an arc always exists by the definition of generalized cycles. Moreover, again by the definition of generalized cycles, the minimization player is not able to leave C. This yields $\nu > 0$.

Given a digraph D representing an instance of (MPG), we construct an instance I of (ES) in the following way. First, we assume without loss of generality that every node in D associated to the minimization player has out-degree one — it is an easy observation that a node with out-degree $q > 1$ can be replaced by q copies with out-degree one without changing the outcome of the game. Moreover, we assume that the weight of the only arc (w, j) leaving a node w associated to the minimization player has weight 0. The case of $d_{wj} \neq 0$ can be handled by replacing $d_{wj} \neq 0$ by 0 and d_{iw} by $d_{iw} + d_{wj}$ for all $i \in in(w)$, recall the transformation in the second paragraph of Section 3.1.

The set V of jobs in the instance I of (ES) is the set of nodes associated to the maximization player. For each node w of the minimization player, we introduce a waiting condition $w = (in_D(w), j)$ where $out_D(w) = \{j\}$. The time lag d_{iw} for $i \in in_D(w)$ is given by the corresponding arc weight in D. Moreover, we add a dummy start node a preceding all other AND-nodes. We refer to this scheduling instance as I'. Finally, in order to obtain the instance I, we modify every waiting condition $w = (X, j)$ with $j \neq j_0$ and $j_0 \notin X$ by adding j_0 to X and setting $d_{j_0 w} = T + 1$; in other words, if job j_0 can be planned, then all other jobs can be planned, too. In particular, instance I is feasible if and only if the earliest start time $S^*_{j_0}$ of job j_0 in instance I' is finite. Thus, it remains to show that $S^*_{j_0} < \infty$ if and only if $\nu \leqslant 0$.

Only if: Consider instance I'. Based on the optimal partial schedule S^* of instance I' ($S_{j_0} < \infty$), we give a strategy for the minimization player which ensures $\nu \leqslant 0$: In each OR-node w with $S^*_w < \infty$, choose an incoming arc (j, w) with $S^*_j + d_{jw} = S^*_w$. As a consequence, $S^*_i < \infty$ for all nodes i visited during the game. Moreover, for two vertices v_1 and v_2 on the path formed by the game, the weight of the (directed) sub-path from v_1 to v_2 is at most $S^*_{v_2} - S^*_{v_1}$. In particular, the length of the cycle terminating the game is at most 0.

If: For an infeasible scheduling instance I, by Lemma 3.2.3, there exists a generalized cycle C in the corresponding digraph D_I which only contains cycles of positive length. Without loss of generality, C contains the node j_0. Notice that C also forms a generalized cycle for the digraph D. We give a strategy for the maximization player which ensures $\nu > 0$: In each step, choose an arc which starts at a node in C. Such an arc always exists by the definition of generalized cycles. Moreover, again by the definition of generalized cycles, the minimization player is not able to leave C. This yields $\nu > 0$. □

With Lemma 3.2.5, it follows from the work of Jurdziński (1998) that the decision problem corresponding to the scheduling problem (ES) is in UP ∩ co-UP. Moreover, (MPG) and hence also (ES) can be computed in sub-exponential time: Zwick and Paterson (1996) have shown that so-called *simple stochastic games* are at least as hard as mean payoff games. The outcome of simple stochastic games can be computed in sub-exponential time, as has been shown by Ludwig (1995). Despite these observations, there is no polynomial time algorithm for (ES) with arbitrary arc weights known and thus, the problem has currently a very rare complexity status, like, the problem PRIMES (decide whether a given natural number is prime).

3.3 Polynomial Algorithms

In the following we give strongly polynomial algorithms of different complexity for the cases $d_{jw} > 0$ and $d_{jw} \geqslant 0$. We propose an $O(|V|+|A|+|\mathcal{W}|\log|\mathcal{W}|)$ algorithm for the restricted model with positive arc weights (see also (Gallo, Longo, Pallottino, and Nguyen 1993)). For the case $d_{jw} \geqslant 0$, which has to be treated in a completely different way, we derive an algorithm of complexity $O(|V| + |A| \cdot |\mathcal{W}|)$. Throughout the discussion of these algorithms we call a job *planned* as soon as its start time has been fixed by the considered algorithm.

3.3.1 Positive Arc Weights

In this section we restrict to the case of positive arc weights or, more general, non-negative arc weights without cycles of length 0 in D. Like Knuth (1977) and Dinic (1990) we basically obtain a slight generalization of Dijkstra's shortest path algorithm.

The algorithm maintains a partial schedule $S \in \mathbb{N}_0^{|\mathcal{V}|}$ where initially $S_w = \infty$ for all OR-nodes w. All currently not planned OR-nodes are maintained in a heap where the sorting key for node w is its tentative start time S_w.

Having set $S_a = 0$ (and also $S_w = 0$ for all $w \in out(a)$) we proceed over time by always choosing an OR-node $w = (X, j)$ with minimum start time from the heap and plan w at its tentative start time S_w. If all other OR-nodes (X', j) preceding j have already been planned, we also plan j at the current time. In this case, the start times of all OR-nodes w' with $w' \in out(j)$ are updated to $S_{w'} := \min\{S_{w'}, S_j + d_{jw'}\}$. If after termination some OR-node w is started at $S_w = \infty$ the considered instance is infeasible. Implementational details are given in Algorithm 3.

If we apply Algorithm 3 to Example 2.2.1 (arc weights are as in Figure 2.1), we obtain the start times $(0, 0, 0, 2, 3, 2, 3)$ for AND-nodes and $(2, 1, 2, 3, 3)$ for OR-nodes. One possible order in which start times get fixed is $1 \prec 2 \prec 3 \prec w_2 \prec w_1 \prec 4 \prec w_3 \prec 6 \prec w_5 \prec 7 \prec w_4 \prec 5$.

Theorem 3.3.1. *For a given set of* AND/OR *precedence constraints represented by a digraph $D = (V \cup \mathcal{W}, A)$ with non-negative arc weights and without cycles of length 0, Algorithm 3 computes an optimal partial schedule S. In particular, the instance is infeasible if and only if $S_w = \infty$ for some* OR*-node w.*

Proof. In this proof we say that an AND-node is *planned* if its start time is fixed (Lines 1 and 3) while an OR-node is planned if it is removed from the heap (Line 2).

By construction of Algorithm 3, S is a feasible partial schedule. Assume that S is not optimal and let v be a node with $S_v > S_v^*$ and S_v^* minimal. If v is an AND-node, then there must exist an OR-node $w = (X, v)$ with $S_w = S_v > S_v^* \geqslant S_w^*$ and we set $v := w$. Otherwise, if v is an OR-node (X, j), then there must exist an AND-node $i \in X$ with $S_v^* = S_i^* + d_{iv}$ and $S_i > S_i^*$. The latter inequality follows from the fact that start times (in the order in which nodes are planned) are non-decreasing. Thus, the choice of v yields $d_{iv} = 0$ and we set $v := i$. Iterating this argument, we can construct a cycle (since there are only finitely many nodes), which has length 0 — a contradiction. □

Lemma 3.3.2. *Algorithm 3 can be implemented to run in* $O(\,|\mathcal{W}| \log |\mathcal{W}| + |A| + |V|\,)$ *time.*

Algorithm 3: Computation of earliest job start times for digraphs without cycles of length 0

Input : A directed graph D representing a set V of jobs and waiting conditions $\mathcal{W}$ with positive arc weights on the arcs in $V \times \mathcal{W}$.

Output: A feasible (partial) schedule $S \in \mathbb{N}_0^{|\mathcal{V}|}$.

$Heap := \emptyset$;
for AND-*nodes* $j \in V$ **do** $a(j) := |in(j)|$;
1 $S_a := 0$; // AND-node a is planned at time 0
for OR-*nodes* $w \in \mathcal{W}$ **do**
  if $w \in out(a)$ **then** insert w in $Heap$ with key $S_w := 0$;
  else insert w in $Heap$ with key $S_w := \infty$;

while $Heap \neq \emptyset$ **do**
2 remove next OR-node $w_0 = (X, j)$ from $Heap$; // OR-node is planned
  reduce $a(j)$ by 1;
  if $a(j) = 0$ **then**
3 $S_j := \max_{w \in in(j)} S_w$; // AND-node is planned
    for OR-*nodes* $w \in out(j)$ **do**
      $S_w := \min\{S_w, S_j + d_{jw}\}$;
      decrease key of w in $Heap$ to S_w;
  delete node w_0 and all incident arcs from D;
return S;

Proof. Since each OR-node enters the heap precisely once the while-loop is executed $|\mathcal{W}|$ times. Each AND-node is planned only once and therefore the inner for-loop is executed at most $|A|$ times. If we choose a Fibonacci-heap for maintaining the OR-nodes, the cost of Line 2 is $\log |\mathcal{W}|$ and we obtain the claimed running time. □

In contrast to previously proposed algorithms, the heap data structure only maintains OR-nodes which leads to the improved running time $O(\,|\mathcal{W}| \log |\mathcal{W}| + |A| + |V|\,)$ instead of $O(\,(|V| + |\mathcal{W}|) \log(|V| + |\mathcal{W}|) + |A|\,)$.

3.3.2 Non-Negative Arc Weights

In extension of the case discussed in Section 3.3.1 we present an $O(\,|V| + |A| \cdot |\mathcal{W}|\,)$ algorithm that is capable to deal with arbitrary arc weights $d_{jw} \geqslant 0$ and thus with cycles of length 0 in D. The problem has independently been considered by Levner, Sung, and Vlach (2000) who observed that the algorithm proposed

by Knuth (1977) fails to compute earliest job start times, when cycles of length 0 occur. Coincidently, and also independently from our work, Adelson-Velsky and Levner (1999) discovered an $O(|A|^2)$ algorithm for the problem. Their basic approach is closely related to our algorithm which is presented next. However, due to an appropriate update of (tentative) start times of OR-jobs we obtain the running time of $O(|V| + |A| \cdot |\mathcal{W}|)$.

Our algorithm is based on the observation that, according to the general feasibility criterion (Lemma 3.2.3), we need to find cycles C in D where each arc of C has weight 0.

A rough scheme of the algorithm is as follows. Analogously to Algorithm 3 we maintain all OR-nodes w in a heap where the sorting key is its tentative start time S_w (initially $S_w = \infty$). Furthermore, whenever an AND-node j is planned, the start times of all OR-nodes $w \in out(j)$ are updated to $S_w = \min\{S_w, S_j + d_{jw}\}$. We proceed over time starting at $t = 0$. For the current time t we compute a set U of (non-started) nodes that can be started at t. U is computed by maintaining the induced subgraph D^0 of D where all planned nodes and all arcs of positive weight have been deleted. In D^0, U is computed as a set of nodes such that for each AND-node j all predecessors $w \in in_{D^0}(j)$ are also in U and for each OR-node w, at least one predecessor $j \in in_{D^0}(w)$ is also in U. Then, as we will prove in Theorem 3.3.3 below, all nodes of U can be started at the current time t. Next we remove a new OR-node w from the heap and increase t to S_w. If $t = \infty$ the algorithm stops. Then, either no OR-node was left in the heap (and we have computed a feasible schedule) or all OR-nodes w in the heap fulfill $S_w = \infty$ (indicating that the given instance is infeasible). Details are provided in Algorithm 4.

If we apply Algorithm 4 to Example 2.2.1 with arc weights as in Figure 2.1 except $d_{5w_1} = 0$ and $d_{4w_4} = 0$ we get:

Iteration 1: $U = \{1, 2, 3\}$, $w = w_2$, $t := 1$

Iteration 2: $U = \{4, 5, w_1, w_4\}$, $w = w_3$, $t := 2$

Iteration 3: $U = \{6\}$, $w = w_5$, $t := 2$

Iteration 4: $U = \{7\}$, $Heap = \emptyset$, $t := \infty$

Thus, we obtain start times $(0, 0, 0, 1, 1, 2, 2)$ for AND-nodes and $(1, 1, 2, 1, 2)$ for OR-nodes.

Theorem 3.3.3. *For a given set of* AND/OR *precedence constraints represented by a digraph $D = (V \cup \mathcal{W}, A)$ with non-negative weights on the arcs, Algorithm 4 computes an optimal partial schedule S. In particular, the instance is infeasible if and only if $S_w = \infty$ for some* OR*-node w.*

Algorithm 4: Computation of earliest job start times for non-neg. time lags

Input : A directed graph D representing a set V of jobs and waiting conditions $\mathcal{W}$ with non-negative arc weights on the arcs in $V \times \mathcal{W}$.

Output: A feasible (partial) schedule $S \in \mathbb{N}_0^{|\mathcal{V}|}$.

```
set D^0 := D and remove all arcs with positive weight from D^0;
t := 0;
Heap := ∅;
for OR-nodes w ∈ W do
    S_w := ∞;
    insert w in Heap with key S_w;
while t < ∞ do
    compute U ⊆ V(D^0) maximal with
1       (in_{D^0}(j) ⊆ U  ∀ j ∈ U∩V) and (in_{D^0}(w)∩U ≠ ∅  ∀ w ∈ U∩W);
    for AND-nodes j ∈ U  (j ∈ U ∩ V)  do
        S_j := t;        // node j is planned at time t
        for OR-nodes w ∈ out_D(j) do
2           S_w := min{S_w, S_j + d_jw};
3           decrease key of w in Heap to S_w;
    for OR-nodes w ∈ U  (w ∈ U ∩ W)  do
4       S_w := t;        // node w is planned at time t
        remove w from Heap;
    Delete all nodes from U in D and D^0;
    if Heap ≠ ∅ then
5       remove the next OR-node w from Heap;
6       t := S_w;
7       remove w from D and D^0;        // node w is planned at time t
    else  t := ∞;
return S;
```

Proof. We first prove that the variable t never decreases, i. e., the algorithm proceeds over time and tries to plan the jobs (and remove them from D and D^0) as early as possible in order of non-decreasing start times. Assume that t decreases in Line 6 of the algorithm and let t_0 denote its value before the decrease. Since the OR-node w determining t was not chosen in Line 5 during the last iteration of the while-loop (when t was set to t_0), S_w has decreased during the current iteration in lines 2 and 3. This is a contradiction to $S_j = t_0$ and $d_{jw} \geqslant 0$.

Observe that the start time S_i of any node $i \in \mathcal{V}$ is never changed after the node is planned (and thus deleted from the graphs D and D^0).

We can now prove that the partial schedule S returned by Algorithm 4 is feasible by verifying all constraints of (ES). By construction of the algorithm, for an AND-node $j \in V$, every OR-node $w \in in(j)$ has either been planned before or is planned together with j in the same iteration of the while-loop; this follows from the first property of U in Line 1. Thus, the constraint in (ES) corresponding to j is fulfilled.

Consider now an arbitrary OR-node $w \in \mathcal{W}$. If w is planned as part of a subset U in Line 4, it follows from the second property of U in Line 1 that there is a job $j \in in(w)$ with $d_{jw} = 0$, and j is planned at the same time as w. Otherwise, if w is planned in Line 7 and $S_w < \infty$, the start time S_w of w must have been decreased in some iteration of the while-loop in Line 2; since the start time S_j of the node $j \in V$ causing the last decrease of S_w has not changed since then, $S_w = S_j + d_{jw}$ in the final partial schedule S. Thus, the constraint in (ES) corresponding to w is fulfilled.

Next we prove that the partial schedule S returned by Algorithm 4 is optimal. Let S^* be the optimal partial schedule and assume that there are nodes $i \in \mathcal{V}$ with $S^*_i < S_i$; we choose such an i' with minimum $S^*_{i'}$ and set $t_0 := S^*_{i'}$; let $U_0 = \{i \in \mathcal{V} \mid t_0 = S^*_i < S_i\}$. We distinguish two cases.

First case: In some iteration of Algorithm 4, t adopts the value t_0. We consider the iteration of the while-loop in which t is increased above t_0 in Line 6. Let D^0 be the digraph at the beginning of the iteration and U the set computed at the start of this iteration. Then $U \cap U_0 = \emptyset$ and, by maximality of U, the set $U \cup U_0$ cannot satisfy the conditions in Line 1. Since S^* is a feasible partial schedule, the first condition of Line 1 is valid for $U \cup U_0$, i. e., $in_{D^0}(j) \subseteq U \cup U_0$ for all $j \in (U \cup U_0) \cap V$. Thus, the second condition is violated: there exists a node $w \in U_0 \cap \mathcal{W}$ with $in_{D^0}(w) \cap (U \cup U_0) = \emptyset$. Moreover, by optimality of S^*, there exists a node $j \in in_D(w)$ with $S^*_w = S^*_j + d_{jw}$, in particular $S^*_j \leqslant t_0$. We next show that $S_j = S^*_j$. If $S^*_j < t_0$ the claim follows from the minimality of t_0. Otherwise, observe that $j \notin U_0$ (if $j \in U_0$ we have $j \in in_{D^0}(w)$ which contradicts $in_{D^0}(w) \cap (U \cup U_0) = \emptyset$). Then, with $S^*_j = t_0$ and $j \notin U_0$ it follows from the definition of U_0 that $S_j = S^*_j$. In particular, S_w has been set to $S_j + d_{jw} = S^*_w$ in Line 2 after j was planned. Since S_w is never increased in Algorithm 4, we get a contradiction to $S_w > S^*_w$.

Second case: t never adopts the value t_0 in Algorithm 4; in particular, $t_0 > 0$ and $U_0 = \{i \in \mathcal{V} \mid S^*_i = t_0\}$. Since S^* is optimal, decreasing all start times S^*_j for $j \in U_0$ to $t_0 - 1$ violates a constraint of (ES). Thus, there exists a node $w \in U_0 \cap \mathcal{W}$ such that $S^*_w = S^*_j + d_{jw}$ for some $j \in V$ with $d_{jw} > 0$, i. e., $S^*_j < t_0$. Therefore, $S_j = S^*_j$ and S_w has been set to $S_j + d_{jw} = S^*_w$ in Line 2 after j was planned. Since S_w is never increased in Algorithm 4, we get a contradiction to $S_w > S^*_w$. □

The bottleneck for the running time of Algorithm 4 is the computation of the set U in each iteration of the while-loop. In fact, it turns out that the linear time algorithm for checking feasibility of a set of AND/OR precedence constraints (for the case of positive arc weights) provides an elegant and fast solution for this problem.

Lemma 3.3.4. *Given a bipartite digraph D with node set $N \cup M$ and arc set A, the (unique) maximal set $U \subseteq N \cup M$ with $in_D(w) \subseteq U$, for all $w \in U \cap N$, and $in_D(j) \cap U \neq \emptyset$, for all $j \in U \cap M$ can be computed in linear time.*

Proof. First, for U and U' fulfilling the conditions given in the lemma, also their union $U \cup U'$ fulfills those conditions. Therefore, such a unique maximal subset U exists.

We show that U can be computed by applying essentially Algorithm 1 to an appropriately constructed instance. Define the set $V = M$ of jobs and the following set $\mathcal{W}$ of waiting conditions: For each $w \in N$ and each $j \in out_D(w)$, introduce a waiting condition $(in_D(w), j)$. Notice that the input size of this instance is not necessarily linear in the input size of the given digraph D since the set $in_D(w)$ is stored once for every $j \in out_D(w)$. We can avoid this undesired increase in the input size by storing, for each $w \in N$, the corresponding waiting conditions as $(in_D(w), out_D(w))$ with the interpretation that every job in the second set is a waiting job for the first set. Algorithm 1 can easily be adapted to handle this compactified input in linear time by replacing the for-loop starting in Line 1 with

for *waiting conditions* $(X, Y) \in \mathcal{W}$ *with* $i \in X$ **do**
  for $j \in Y$ **do**
    decrease $a(j)$ by 1;
    if $a(j) = 0$ **then** add j to Q;
  remove (X, Y) from $\mathcal{W}$;

By Corollary 2.3.3, Algorithm 1 computes a set $L \subseteq V$ such that $V' := V \setminus L$ is a maximal subset of V with the following property: For all $j \in V'$ there exists a waiting condition $(X, j) \in \mathcal{W}$ with $X \subseteq V'$. Thus, the set

$$U = \big(\{w \in N \mid in(w) \subseteq V'\} \cup V'\big) \subseteq N \cup M$$

fulfills the conditions given in the lemma, i. e., $in_D(w) \subseteq U$, for all $w \in U \cap N$, and $in_D(j) \cap U \neq \emptyset$, for all $j \in U \cap M$. Assume that there is a bigger set $U^* \supset U$ that also fulfills these conditions. By construction of U, there exists a node $j \in M \cap (U^* \setminus U)$. Since the set $U^* \cap M$ of jobs has the property described in Corollary 2.3.3, we get a contradiction to the maximality of V'. □

With the help of this lemma, we can now give a bound on the running time of Algorithm 4.

Corollary 3.3.5. *Algorithm 4 can be implemented to run in* $O(|\mathcal{W}| \cdot |A| + |V|)$ *time.*

Proof. First, all isolated AND-nodes are planned and thus removed from D^0 in the first iteration of the while-loop. Moreover, in each iteration, at least one OR-node is removed from D^0 and the number of iterations is thus bounded by $|\mathcal{W}|$. Finally, the running time of each iteration is dominated by the computation of U which can be done in $O(|A|)$ time. □

Notice that, in the sense of Lemma 3.3.4, Algorithm 4 and its worst case complexity (Corollary 3.3.5) are both valid for digraphs D where OR-jobs have multiple outgoing arcs (of length 0).

3.4 The Linear Time-Cost Tradeoff Problem

We conclude the chapter by studying the *linear time-cost tradeoff problem* with AND/OR precedence constraints imposed among jobs. In the time-cost tradeoff problem the processing times of jobs depend on the amount of money (or general resource) that is paid for it. For each job j, this dependence is described by a non-increasing, non-negative, affine linear *cost function* $c_j : [p_j^{min}, p_j^{max}] \to \mathbb{R}_{\geqslant}$, where $p_j^{min} \geqslant 0$ refers to the smallest possible processing time of j and $p_j^{max} \geqslant p_j^{min}$ is the largest possible processing time of j. The value $c_j(p_j)$ is the amount of money one has to pay to run j with processing time p_j.

The linear time-cost tradeoff problem was formulated more than 40 years ago by Kelley and Walker (1959): The problem is to find for all deadlines $T \geqslant 0$ the minimal total cost $c(p) = \sum_{j \in V} c_j(p_j)$ of the jobs such that $C_{max}(p) \leqslant T$. Recall that $C_{max}(p)$ denotes the project makespan with respect to some vector p of processing times. The solution thus consists of a function (the so-called *time-cost tradeoff curve*) $B(T) := \min\{c(p) | p^{min} \leqslant p \leqslant p^{max}, C_{max}(p) \leqslant T\}$. For traditional precedence constraints, $B(T)$ is piecewise linear, convex, non-increasing, and continuous, as was discovered independently by Fulkerson (1961) and Kelley (1961). They showed that $B(T)$ can be constructed by a series of minimum cut computations. Later, Phillips and Dessouky (1977) suggested an improved version of their algorithms.

Let us discuss the curve $B(T)$ when AND/OR precedence constraints are imposed among jobs. Throughout the discussion we assume that the given set of constraints is feasible in the sense of Lemma 2.3.2. We show that $B(T)$ is still non-increasing and piecewise linear, but not continuous (and hence not convex)

anymore. Recall that each realization defines a time-cost tradeoff curve for a set of traditional precedence constraints. Since we may choose any realization R of $\mathcal{W}$, $B(T)$ is the minimum of a set of traditional time-cost tradeoff curves. As such, it is no longer continuous but fulfills the other above mentioned properties. We next present an alternative argumentation that is based on a polyhedral view to the problem. We here follow an idea of Skutella (1998, Lemma 1.2.1). He gave a new proof for the original result of Fulkerson (1961) and Kelley (1961) on the above mentioned properties of $B(T)$ for traditional precedence constraints. Consider the set $P \subset \mathbb{R}^{2n+2}_{\geqslant}$ defined by the points $(T \in \mathbb{R}_{\geqslant}, B \in \mathbb{R}_{\geqslant}, p \in \mathbb{R}^n_{\geqslant}, S \in \mathbb{R}^n_{\geqslant})$ that fulfill $S_j \geqslant 0$, $p_j^{min} \leqslant p_j \leqslant p_j^{max}$ for all $j \in V$, and

$$\begin{aligned} \bigvee\nolimits_{i \in X} (S_j \geqslant S_i + p_i) \qquad & (X, j) \in \mathcal{W}, \\ T \geqslant S_j + p_j \qquad & j \in V, \\ B \geqslant \sum\nolimits_{j \in V} c_j(p_j) \ . & \end{aligned}$$

Since $c(p)$ is affine linear, P is the union of finitely many polyhedra, one for each realization. If we project P on the coordinates (T, B) we obtain the set $P' \subset \mathbb{R}^2_{\geqslant}$ with $P' := \{(T, B) |$ there exists $(T, B, p, S) \in P\}$. Clearly, P' is again the union of finitely many polyhedra. By definition of $B(T)$ it follows that $P' = \{(T, B) | B \geqslant B(T)\}$. Hence, the border of P' defines $B(T)$. Consequently, $B(T)$ is piecewise linear. We next explain the above argumentation by an example which also shows that $B(T)$ is not continuous.

Example 3.4.1. *Let $V = \{1, 2, 3\}$ and let $(\{1, 2\}, 3)$ be the only waiting condition. Furthermore, let $p_1^{min} = 0$, $p_1^{max} = 3$, $p_2^{min} = 1$, $p_2^{max} = 2$, $p_3^{min} = 3$, and $p_3^{max} = 3$. The cost functions for the jobs 1 and 2 are defined by $c_1(p_1) = -\frac{2}{3}p_1 + 2$, $c_2(p_2) = -p_2 + 2$, and $c_3(p_3) = 0$. Notice that p_3 is fixed.*

Figure 3.1 shows P' for Example 3.4.1. P' is the union of two polyhedra each of which corresponds to one realization. The time-cost tradeoff curve $B(T)$ equals the highlighted (bold) part of the border of P'. We see that $B(T)$ is not continuous at $T = 4$. The reason is that job 2, which precedes job 3 for $T \in [4, 6]$, cannot be further shortened. We therefore change to the realization where job 3 waits for job 1 ($T \in [3, 4)$).

Let us summarize the outcome of Example 3.4.1 and the above argumentation in the following lemma.

Lemma 3.4.2. *If* AND/OR *precedence constraints are imposed among jobs, then the time-cost tradeoff curve $B(T)$ is non-increasing and piecewise linear. In general, it is not continuous (and thus not convex).*

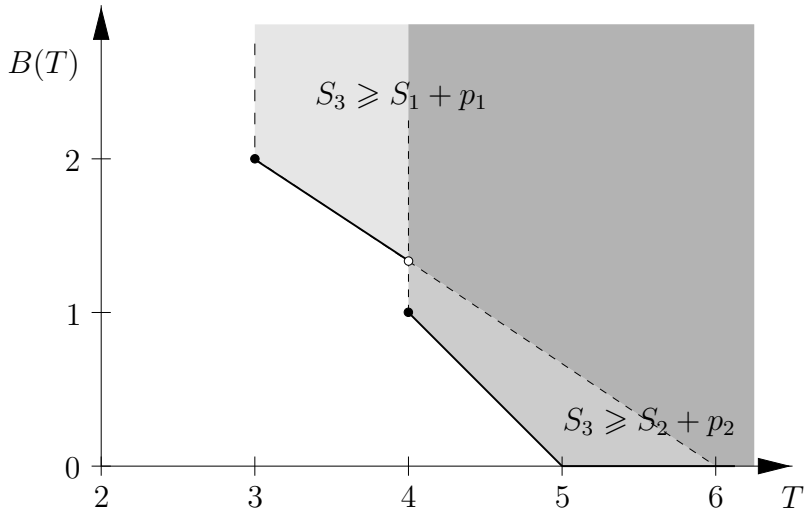

Figure 3.1: The figure shows the set P' and the time-cost tradeoff curve $B(T)$ (bold) for Example 3.4.1.

As a consequence, one may guess that computing $B(T)$ is much harder in the presence of AND/OR precedence constraints when compared to traditional precedence constraints. In the following we show that the linear time-cost tradeoff problem is strongly NP-hard for AND/OR precedence constraints. In fact, for the reduction which is presented next, it suffices to consider a special case of the linear time-cost tradeoff problem, namely the so-called

> DEADLINE PROBLEM: For a given deadline $T \geqslant 0$ and an integer budget b, do there exist processing times p satisfying $C_{max}(p) \leqslant T$ and $c(p) \leqslant b$.

We show the NP-completeness of the DEADLINE PROBLEM with AND/OR precedence constraints by a reduction from HITTING SET.

> HITTING SET: Let V be a finite set, $\mathcal{F}$ be a collection of subsets of V, and let b be a positive integer. Does there exist a subset $V' \subseteq V$ that contains at least one element from each set in $\mathcal{F}$ and $|V'| \leqslant b$.

Recall that the HITTING SET problem is NP-complete in the strong sense, see, e. g., (Garey and Johnson 1979, SP8). We obtain the following theorem.

Theorem 3.4.3. *The* DEADLINE PROBLEM *with* AND/OR *precedence constraints is NP-complete in the strong sense.*

Proof. The problem is obviously in NP; a polynomially checkable proof is p itself. Algorithm 4 (which was formulated for $p_j \in \mathbb{N}$ but also works for $p_j \in \mathbb{R}_{\geqslant}$) delivers the project makespan $C_{max}(p)$.

Given an instance of HITTING SET, we introduce a job for each element of the ground set and one additional job j. The processing time of j is fixed to $p_j = 1$, and all other jobs $i \neq j$ have $p_i^{min} = 0$ and $p_i^{max} = 1$. Moreover, all cost functions are defined by $c_i(p_i) = -p_i + 1$. Now, for each subset $X \in \mathcal{F}$ of the HITTING SET problem, introduce a waiting condition (X, j). We then ask whether there exist processing times p such that the project is completed at time $T = 1$ and the total cost is less than or equal to b. We show that the so-defined special case of the DEADLINE PROBLEM is equivalent to the HITTING SET problem. To keep notation simple we use the same notation for both problems. Suppose that the answer to the instance of the HITTING SET problem is 'yes'. Then, there exists a set $V' \subseteq V$ with cardinality less than or equal to b such that V' contains at least one element from each set in $\mathcal{F}$. If we set the processing time of each job in V' to 0 and all others to 1, by construction of the instance of the DEADLINE PROBLEM, $S_j = 0$ is feasible and the cost $c(p) = |V'|$. Thus, the answer to the DEADLINE PROBLEM is 'yes'. Now suppose that the answer to an instance of the special case of the DEADLINE PROBLEM is 'yes'. Then, by definition of the cost functions c_j, the set V' of jobs which have processing time 0 is of cardinality $|V'| \leqslant b$. Since j can be started at $S_j = 0$ we have that V' contains at least one job from each of the sets $X \in \mathcal{F}$. Consequently, the answer to the HITTING SET problem is 'yes'. □

We finally consider the minimization problems that correspond to the DEADLINE PROBLEM and the HITTING SET problem. The minimization problem which corresponds to the DEADLINE PROBLEM is to find, for given $T \geqslant 0$, processing times p such that $C_{max}(p) \leqslant T$ and $c(p)$ is minimal. The minimization problem which corresponds to the HITTING SET problem is to find a set $V' \subseteq V$ of minimum cardinality such that V' contains at least one element from each set in $\mathcal{F}$. It is NP-hard to approximate the latter problem within $\epsilon \log |V|$ for some $\epsilon > 0$. This fact follows from the polynomial equivalence of the HITTING SET problem and the SET COVERING problem (Ausiello, d'Atri, and Protasi 1980). The above mentioned inapproximability result was established for the SET COVERING problem by Raz and Safra (1997); one may alternatively choose other inapproximability results for SET COVERING or HITTING SET. We have shown that each instance of the above defined special case of the DEADLINE PROBLEM with cost $c(p) = b$ can be transformed into an instance of the hitting set problem with cost $|V'| = b$ and vice versa. As a consequence, the inapproximability results for SET COVERING and HITTING SET carry over to the minimization problem which corresponds to the DEADLINE PROBLEM with AND/OR precedence constraints.

CHAPTER 4

REPRESENTATION OF RESOURCE CONSTRAINTS IN PROJECT SCHEDULING

Already in Chapter 1 we noted that, in project scheduling, resource constraints are usually defined via resource consumption and availability. Many algorithmic approaches, however, are based on a different concept, the so-called *minimal forbidden sets* to represent the resource constraints. Our interest in minimal forbidden sets relies on the fact that they play an important role in the context of *scheduling policies* for stochastic resource-constrained project scheduling; we extensively study this topic in Chapter 5 below. In this chapter, we discuss the connection between both representations of resource constraints which reveals a close relation to *threshold hypergraphs*. In addition, for given resource consumption and availability, we propose a simple backtracking algorithm to efficiently compute and represent all minimal forbidden sets. We computationally evaluate the algorithm on test sets of the project scheduling problem library PSPLIB. The chapter is based on joint work with Marc Uetz (Stork and Uetz 2000).

4.1 Introduction

In the previous two chapters we established results that are related to *precedence constraints* among jobs. As another step towards the stochastic resource-constrained project scheduling problem we now focus on *resource constraints* and their representation. To this end, we consider a model where both precedence constraints and resource constraints have to be respected. The results presented are completely independent of job processing times, and are thus applicable to both deterministic and stochastic models.

Let us briefly recall the classical representation of resource constraints. Jobs need different (renewable) resource types $k \in K$ while being processed. A constant amount of $R_k \in \mathbb{N}$ units of each resource type is available throughout the project and each job j consumes $0 \leqslant r_{jk} \leqslant R_k$ ($r_{jk} \in \mathbb{N}$) units of resource $k \in K$ while in process. We call this representation the *threshold representation* (the motivation for this notation will become clear in Section 4.2). The other representation of resource constraints, which we call the *(minimal) forbidden set*

representation, is as follows. A subset $F \subseteq V$ of jobs is called *forbidden* if the jobs in F are an anti-chain of the partial order defined by the precedence constraints E_0, and the total resource consumption $\sum_{j \in F} r_{jk}$ exceeds the resource availability R_k for some $k \in K$. F is called *minimal forbidden* if any proper subset $F' \subset F$ is *resource-feasible*, that is, $\sum_{j \in F'} r_{jk} \leqslant R_k$ for all $k \in K$. Let us denote by $\mathcal{F}$ the system of minimal forbidden sets. To give an example, recall Example 1.3.1 and observe that $\mathcal{F}$ consists of the three sets $\{1, 5\}$, $\{2, 3, 4\}$ and $\{2, 4, 5\}$.

Minimal forbidden sets are an important concept to represent resource constraints. In fact, they form the basis of numerous algorithmic approaches to resource-constrained project scheduling. Probably the most important field of application is stochastic scheduling, the topic of the thesis. We discuss this topic in the following Chapters 5 and 6. Minimal forbidden sets also play a role in algorithmic approaches to deterministic project scheduling problems, e. g. in (Radermacher 1985; Bartusch, Möhring, and Radermacher 1988). In addition, forbidden sets are useful to derive cutting planes within integer programming approaches, e. g. in (Alvarez-Valdés Olaguíbel and Tamarit Goerlich 1993; Möhring, Schulz, Stork, and Uetz 2000). Recall from Chapter 1 that Schäffter (1997) discusses inapproximability results for resource-constrained project scheduling problems by means of the forbidden set representation of resource constraints. (He argued that vertex coloring in undirected graphs reduces to scheduling subject to forbidden sets). Finally, forbidden sets can be seen as a generalization of the *disjunctive graph* concept known from shop scheduling, as pointed out by Radermacher (1985); see also (Balas 1971). In shop scheduling problems, each minimal forbidden set consists of exactly two jobs.

This chapter is organized as follows. In Section 4.2, we start by discussing theoretical issues related to the threshold and minimal forbidden set representations of resource constraints, revealing a close relation to threshold (hyper-)graphs. In Section 4.3, we propose a backtracking algorithm which computes the system $\mathcal{F}$ of minimal forbidden sets for a scheduling instance which is given by the usual threshold representation. We show that, for instances with only one resource type ($|K| = 1$), the algorithm can be implemented to run in polynomial time with respect to the in- and output. A computational evaluation of the algorithm is presented in Section 4.4, based on the instances from the project scheduling library PSPLIB (2000). The results exhibit the benefits of the proposed algorithm in comparison to an approach to compute $\mathcal{F}$ previously suggested by Bartusch (1984). Our results also provide further insights in the structure of the instances of the library. We conclude with some remarks and examples in Section 4.5.

4.2 Threshold and Forbidden Set Representations

In this section, we address several questions which are related to the transformation between the two above mentioned representations of resource constraints.

4.2.1 Relations to Threshold (Hyper-)Graphs

One can think of the system of minimal forbidden sets $(V, \mathcal{F})$ as an undirected hypergraph where jobs of the scheduling instance correspond to the vertices of the hypergraph and the minimal forbidden sets correspond to hyperedges. Let us first address the question if these hypergraphs have any particular property. To start with, consider the following problem: Given a problem instance with a minimal forbidden set representation of the resource constraints, what is the minimal number of resource types k required in a threshold representation that represents $\mathcal{F}$? Obviously, $|\mathcal{F}|$ different resource types suffice, because one may introduce one resource type k for each minimal forbidden set $F \in \mathcal{F}$ with $R_k = |F| - 1$ and $r_{jk} = 1$ for each $j \in F$ and $r_{jk} = 0$, otherwise. Moreover, it is easy to see that one resource type does not suffice in general, e. g. with $V = \{1, 2, 3, 4\}$, $E_0 = \emptyset$, and $\mathcal{F} = \{\{1, 2\}, \{3, 4\}\}$. In fact, as will be demonstrated in Example 4.5.1 in Section 4.5, the number of resource types required in a threshold representation can be exponential in n, the number of jobs.

It turns out that a related problem has been studied in the context of *threshold (hyper-)graphs*: According to Golumbic (1980), a threshold hypergraph is an undirected hypergraph $(V, \mathcal{F})$, $\mathcal{F} \subseteq 2^V$, with the following property: A non-negative integer value r_j can be assigned to each vertex $j \in V$ such that there is an integer *threshold* R with the property that a subset $B \subseteq V$ is *stable* if and only if $\sum_{j \in B} r_j \leqslant R$. Here, a *stable set* of a hypergraph is a subset $B \subseteq V$ which does not contain any hyperedge, that is, $F \not\subseteq B$ for all $F \in \mathcal{F}$; see, e. g., (Duchet 1995). In other words, the system of stable sets of a threshold hypergraph can be represented by only one linear inequality, namely $\sum_{j \in V} r_j \, x_j \leqslant R$. Here, $x = (x_1, \ldots, x_n)$ is the characteristic vector of a subset X of V, where $x_j = 1$ if $j \in X$ and $x_j = 0$ otherwise. Notice that the stable sets exactly correspond to the resource-feasible sets in our application, hence $(V, \mathcal{F})$ defines a threshold hypergraph exactly if one resource type suffices to represent the resource constraints (and $E = \emptyset$). In analogy with the definitions for ordinary graphs, the *threshold dimension* t of a hypergraph $(V, \mathcal{F})$ can be defined as the minimum number of inequalities which are required to represent the system of stable sets; see (Chvátal and Hammer 1977). More precisely, there exist t inequalities $\sum_{j \in V} r_{jk} \, x_j \leqslant R_k$, $k = 1, \ldots, t$, such that X is a stable set in $(V, \mathcal{F})$ if and only if all t inequalities are fulfilled. But even for ordinary graphs, the determination of the threshold dimension is NP-hard (Chvátal and Hammer 1977). According to Yannakakis (1982),

already the decision problem if the threshold dimension of a graph is bounded by 3 is NP-complete (the decision problem if the threshold dimension of a graph is bounded by 2 can be solved in polynomial time). We refer to the surveys by Mahadev and Peled (1995) and Brandstädt, Le, and Spinrad (1999) for more details and references. From the above discussion we obtain the following theorem.

Theorem 4.2.1. *Given a project scheduling problem with minimal forbidden set representation $\mathcal{F}$ of resource constraints, and given that the number of minimal forbidden sets $\mathcal{F}$ is polynomial in n, it is NP-hard to determine the minimum number of resource types required in a threshold representation.*

Proof. The claim even holds if all minimal forbidden sets $F \in \mathcal{F}$ have cardinality 2. Then $(V, \mathcal{F})$ is an ordinary graph, and the problem corresponds to the determination of the threshold dimension of that graph, which is NP-hard. □

4.2.2 From Thresholds to Minimal Forbidden Sets

We now discuss the complexity of the computation of $\mathcal{F}$, given the (usual) threshold representation of resource constraints. Clearly, since $\mathcal{F}$ can be exponential in n, the number of jobs, there is no algorithm with polynomial running time with respect to n. However, if only one resource type is present ($|K| = 1$), the following will be proved in Section 4.3.3.

Theorem 4.2.2. *Given a project scheduling problem with threshold representation of resource constraints, and given that the number of resource types $|K|$ equals 1, the minimal forbidden sets $\mathcal{F}$ can be computed in time polynomial in $|\mathcal{F}|$ and n.*

Let us next consider the following three related problems that are important if an instance with threshold representation is given. For a given subset $W \subseteq V$ of jobs, which is an anti-chain of $G_0 = (V, E_0)$, we ask whether

(i) W is minimal forbidden,

(ii) W is contained in a (not necessarily minimal) forbidden set $F \supseteq W$, and

(iii) W is contained in a minimal forbidden set $F \supseteq W$.

It is trivial to decide Problem (i): W must be a forbidden set, that is, $\sum_{j \in W} r_{jk} \geqslant R_k + 1$ for some $k \in K$, and W is minimal forbidden if and only if $W \setminus \{j\}$ is resource-feasible for each $j \in W$ and all k. This can obviously be verified in $O(\,|K||W|\,)$ time. We can also decide Problem (ii) in polynomial time: Denote by $N \subseteq V$ all jobs in V which are unrelated to all jobs in W (with respect to the

precedence constraints). Obviously, if there is a forbidden set F with $W \subseteq F$, then $F \subseteq W \cup N$. In addition, there must be at least one resource type $k \in K$ such that the weight of a maximum weight anti-chain in the partially ordered set that is induced from G_0 on the jobs $W \cup N$ exceeds R_k. A maximum weight anti-chain of a partially ordered set equals a maximum weight stable set in the underlying comparability graph. For each resource type k, this problem can be solved in time polynomial in n as a minimum flow problem; see (Möhring 1985). Finally, Problem (iii) turns out to be NP-complete, since already the following, restricted problem is NP-complete.

Theorem 4.2.3. *Given a project scheduling problem (even without precedence constraints) with threshold representation of the resource constraints, and given that the number of resource types is polynomial in n, it is NP-complete to determine if a given job is contained in some minimal forbidden set $F \in \mathcal{F}$ or not.*

Proof. The problem is obviously in NP; according to the preceding remarks, a polynomially checkable proof is the set F itself. We will use a simple reduction of the NP-complete problem PARTITION (see, e. g., (Garey and Johnson 1979)). The problem PARTITION is the following: We are given n items of integral weight $r_j > 0$ with $\sum_{j=1}^{n} r_j$ even, and the question is if there exists a partition of the items into two subsets of equal total weight. Now define a project scheduling problem as follows. We have no precedence constraints and one job per item, each with resource requirement r_j. The resource availability is $R = \frac{1}{2} \sum_{j=1}^{n} r_j$. In addition, we have one more job, say i, with resource requirement $r_i = 1$. Now, if i is contained in a minimal forbidden set F, we have $\sum_{j \in F \setminus \{i\}} r_j = R$, since F is minimal forbidden and since i requires only one resource unit. On the other hand, if i is not contained in a minimal forbidden set, there is no subset F of the original items with total weight R. This completes the proof. □

4.2.3 Related Topics

Interestingly, threshold graphs and related questions have been considered also in the context of the so-called *PV-chunk synchronizing primitive*, which generalizes the classical *semaphore* concept for synchronization of parallel processing. In fact, apparently prior to Chvátal and Hammer (1977), threshold graphs have been defined and characterized in this context by Henderson and Zalcstein (1977); see also (Ordman 1987).

In the form of *minimal covers*, minimal forbidden sets also arise in the context of the *knapsack polytope*, or more generally knapsack inequalities in 0–1 integer programming. Given a 0–1–polytope $P = \{x \in \{0,1\}^{|V|} \mid \sum_{j \in V} r_j x_j \leqslant R\}$, a *cover* is a set $C \subseteq V$ with $\sum_{j \in C} r_j > R$, and C is called *minimal* if it is minimal

with respect to this property. In other words, minimal covers exactly correspond to minimal forbidden sets in our application. In the context of integer programming, minimal covers play an important role, since they give rise to cover inequalities of the form $\sum_{j \in C} x_j \leqslant |C| - 1$, which are valid for P, and also to lifted cover inequalities, which are even facet-inducing for P. We refer to (Balas and Zemel 1978) for more details.

4.3 Computing Minimal Forbidden Sets

In this section, we propose an algorithm which computes the minimal forbidden set representation $\mathcal{F}$ for an instance which is given in threshold representation. Notice that exponentially many minimal forbidden sets may exist, hence the output of such an algorithm may be exponential with respect to n, the number of jobs. Before we describe the algorithm to enumerate all minimal forbidden sets we discuss the related problem of determining the number of minimal forbidden sets (without writing them down explicitly).

4.3.1 Counting Minimal Forbidden Sets

Let Σ^* be the set of all finite strings over the alphabet Σ. A function $f : \Sigma^* \to \mathbb{N}$ is in #P if there is a non-deterministic polynomial-time bounded Turing machine M that decides some language over Σ^* such that the number of accepting computations of M on input x is $f(x)$, for all $x \in \Sigma^*$. A counting problem, i.e., a function $f : \Sigma^* \to \mathbb{N}$, is said to be #P-complete if $f \in$ #P and every function in #P is polynomial-time reducible to f. For background information on #P we refer to one of the standard texts, e. g., (Garey and Johnson 1979; Papadimitriou 1994). The following theorem is a direct consequence of the work of Provan and Ball (1983).

Theorem 4.3.1. *Given a project scheduling problem with threshold representation of resource constraints, the problem of determining the number of minimal forbidden sets is #P-complete.*

Proof. The problem is easily seen to be in #P since every minimal forbidden set can be recognized in time polynomial in n (see Section 4.2.2). We next give a (parsimonious) reduction from the problem MAXAC. MAXAC is the problem to determine the number of maximum cardinality anti-chains of a partially ordered set. MAXAC was shown to be #P-complete by Provan and Ball (1983).

Given an instance of MAXAC we construct a project scheduling problem with resource constraints as follows. First, compute the cardinality m of a maximum cardinality anti-chain (this can be done in polynomial time, see Section 4.2.2).

Then introduce for each element j of the partially ordered set a job j, and for each ordered pair (i, j) of the partially ordered set a precedence constraint between the jobs i and j. To keep notation simple, we use the same identifiers for jobs and for elements of the partially ordered set. The scheduling instance has one resource type with availability $R = m - 1$. Each job j requires one unit of that resource type ($r_j = 1$). Obviously, there is a one-to-one correspondence between the minimal forbidden sets in the so-constructed scheduling instance and the anti-chains of cardinality m. □

4.3.2 Description of the Algorithm

We now describe our algorithm to compute all minimal forbidden sets for an instance which is given in threshold representation. The basic approach is to enumerate subsets of V in a tree T where each node w of T, except the root node, is associated to exactly one job $j \in V$ (however, the mapping of nodes to jobs is not an injection). If node w is associated to some job j, w has a child node for each job $i = j + 1, \ldots, n$. The root node has a child node for each job $i \in V$. Each node w of the tree defines a subset $W \subseteq V$ of jobs with $j \in W$ by traversing the tree from w to the root node, and collecting the associated jobs on that path. In fact, a node of the tree only consists of its associated job j, a pointer to its father, and, for technical reasons, the (current) number of child nodes. With these basic definitions, there is a one-to-one correspondence between the set of nodes of T and the power set 2^V of all subsets of V.

To build a tree $T(\mathcal{F})$ which exactly represents all minimal forbidden sets $\mathcal{F}$, the tree T is pruned during this generic process: A node w is discarded as soon as it can be proved that neither W nor any superset of W that is located in the subtree rooted at w is a minimal forbidden set. $T(\mathcal{F})$ is constructed in a DFS fashion. For each node w that is to be added within the construction of $T(\mathcal{F})$, it is tested whether W is a minimal forbidden set. This is done in two steps. First, we check whether the associated set W is an anti-chain with respect to the (transitively implicit) precedence constrains. If this is not the case, by definition of minimal forbidden sets, the subtree rooted at w can be discarded (including w itself). Otherwise, if W is forbidden, that is, $\sum_{j \in W} r_{jk} > R_k$ for some $k \in K$, we test whether W is *minimal* forbidden. This is done by verifying whether each set $W \setminus \{j\}$, $j \in W$, is resource-feasible; see Problem (i) in Section 4.2.2. If this is the case then W is minimal forbidden and w is stored as a leaf of the tree $T(\mathcal{F})$. Otherwise, W is not minimal forbidden and the subtree rooted at w can be discarded (including w itself). If W is resource-feasible, there may exist minimal forbidden sets $F \supset W$ that are located in the subtree rooted at w; hence branching is required on w. Finally, if some node does not represent a minimal forbidden

set, and does not have any further descendants, it is deleted from the tree. Notice that deletion of nodes is meant recursively, that is, if a deleted node w was the only child of its father u in $T(\mathcal{F})$ then u is deleted as well. Upon termination, the constructed tree $T(\mathcal{F})$ has precisely $|\mathcal{F}|$ leaves. Figure 4.1 depicts the trees T and $T(\mathcal{F})$ that result from Example 1.3.1.

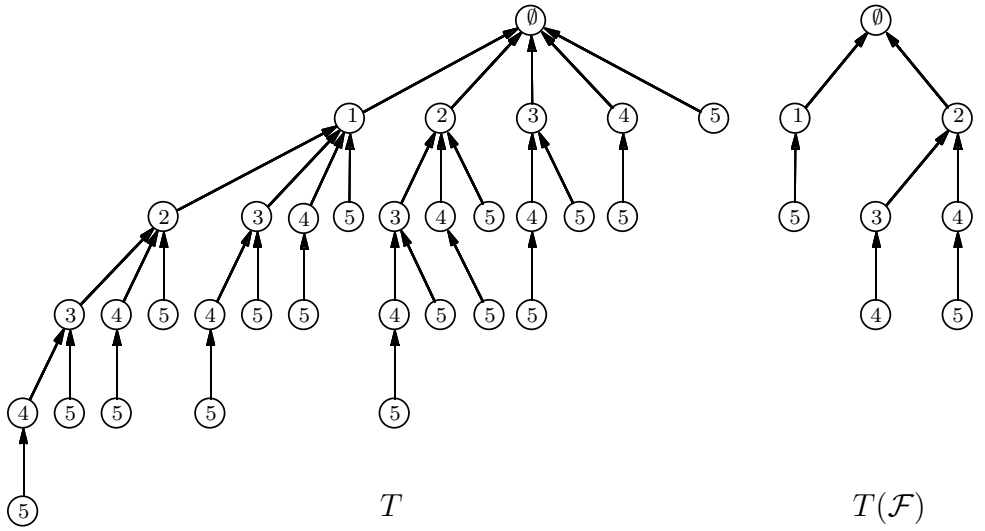

Figure 4.1: The trees T and $T(\mathcal{F})$ for Example 1.3.1.

4.3.3 Analysis of the Algorithm

Let us now discuss the computational complexity of the proposed algorithm. We prove that the algorithm can be implemented to run polynomial in n and $|\mathcal{F}|$, the size of the in- and output, if there is only one resource type (see Theorem 4.2.2 above). Note that, for practical purposes, our implementation differs from the algorithm described next; this will be discussed in Section 4.3.4 below.

We first assume that $|K| = 1$, let $r_j := r_{j,1}$ be the resource consumption of job $j \in V$. To make the above generic procedure polynomial in the size of the in- and output, we consider the jobs in a non-increasing order of their resource consumption r_j; so assume w.l.o.g. that $r_1 \geqslant r_2 \geqslant \cdots \geqslant r_n$. Then the following observation is immediate.

Lemma 4.3.2. *If $|K| = 1$ and if the jobs are considered in non-increasing order of r_j, i. e., in the order $1 \prec 2 \prec \cdots \prec n$ in $T(\mathcal{F})$, then each forbidden set F found by the generic procedure described in Section 4.3.2 is already minimal forbidden.*

Proof. Say a forbidden set $F = \{j_1, j_2 \ldots, j_t\}$ is found, where $j_1 < j_2 < \cdots < j_t$. Then, by construction, the set $F \setminus \{j_t\}$ is resource-feasible, and since $r_{j_1} \geqslant r_{j_2} \geqslant \cdots \geqslant r_{j_t}$, also all sets $F \setminus \{j_i\}$ are resource-feasible for all $i = 1, \ldots, t-1$. □

Hence, the above described procedure only generates nodes w which correspond to anti-chains W which are either resource-feasible or minimal forbidden. Now recall that for any given resource-feasible subset of jobs $W \subseteq V$, which is an anti-chain with respect to the precedence constraints, one can decide in time polynomial in n if W is contained in some (not necessarily minimal) forbidden set or not; this is Problem (ii) mentioned in Section 4.2.2. In particular, at any node w considered in the generic procedure described in Section 4.3.2, associated to some job j, one can decide in time polynomial in n if the corresponding anti-chain W is contained in some forbidden set F with $F \subseteq W \cup \{j+1, \ldots, n\}$ or not. In other words, one can decide in time polynomial in n if node w will eventually lead to some forbidden set or not. Combined with Lemma 4.3.2, we now obtain the following.

Lemma 4.3.3. *If $|K| = 1$ and if the jobs are considered in non-increasing order of r_j, i. e., in the order $1 \prec 2 \prec \cdots \prec n$ in $T(\mathcal{F})$, at any node w considered in the generic procedure described in Section 4.3.2, one can decide in time polynomial in n if w will eventually lead to some minimal forbidden set or not.*

Since the number of nodes in $T(\mathcal{F})$ is obviously polynomial in $|\mathcal{F}|$, this shows that for $|K| = 1$, the time required to compute the tree $T(\mathcal{F})$ is in fact polynomial in n and $|\mathcal{F}|$, which concludes the proof for Theorem 4.2.2.

For $|K| > 1$, however, the described algorithm is not polynomial in $\mathcal{F}$. The reason is that, given a node w considered in the generic procedure described in Section 4.3.2, we can no longer decide in polynomial time whether the associated anti-chain W is contained in a *minimal* forbidden set or not (recall Theorem 4.2.3). This was possible for the case $|K| = 1$ only due to Lemma 4.3.2, which does no longer hold if $|K| > 1$. Consequently, if $|K| > 1$, one possibly ends at nodes w such that the associated set of jobs W is forbidden, but not minimal forbidden. In fact, the number of such nodes may be exponential in $|\mathcal{F}|$ for our algorithm, as is demonstrated by Example 4.5.2 given in Section 4.5.

4.3.4 Implementation and Fast Reduction Tests

Contrary to what was described in Section 4.3.3, in our actual implementation we only considered heuristic but very efficient ‘reduction tests’ in order to decide if a node of the tree potentially leads to a minimal forbidden set or not. These simple tests greatly improved the performance of the simple generic procedure

described in Section 4.3.2; they will be described in this section. To simplify notation, we omit the resource index k. Resource requirements r_j and supply R are treated as vectors and any inequality involving r_j or R is meant componentwise. Moreover, r_W denotes the vector of total resource consumption of W.

First, motivated by the results of Section 4.3.3, also for instances with more than one resource type it showed to be computationally more effective to consider the jobs in a suitable ordering: Therefore we identify a resource $k^* \in K$ that is scarcest, defined as a resource with smallest ratio $R_k / \sum_{j \in V} r_{jk}$, and assume that jobs are numbered in non-increasing order of their consumption of this resource type k^*. Although this does not help theoretically, it helps to heuristically close the gap between r_W and R in as many nodes w as soon as possible. Notice that this is particularly important since for any node w of the generic tree T, the subtree rooted at w is extremely unbalanced: If $s(j)$ denotes the size of a subtree rooted at some node w associated to job j, then $s(j) = 1 + \sum_{k=j+1}^{n} s(k)$, hence $s(j) = 2^{n-j}$.

Next, two jobs i and j cannot be in a common forbidden set if there is a (transitively implicit) precedence constraint between i and j. In addition, we implemented two other heuristic tests to determine if no minimal forbidden set contains both i and j. First, if the resources required by i and j are *disjoint* in the sense that $r_{ik} \cdot r_{jk} = 0$ for each $k \in K$, then i and j together do not belong to any minimal forbidden set. Second, let U be the set of jobs that are unrelated to both i and j with respect to the (transitively implicit) precedence constraints. Then, if $r_j + r_i + r_U \leqslant R$, then i and j are not contained in a common minimal forbidden set, either. All above tests are performed as preprocessing, and the resulting information is stored in a Boolean matrix M of size $n \times n$ in order to provide access in O(1) time.

Finally, we implemented another heuristic test which is particularly useful to keep the tree small if resource constraints are weak; it is a heuristic test in order to detect nodes w which cannot lead to any forbidden set: For a given node w, associated to some job j and a set W of jobs, $j \in W$, we simply sum up the resource requirements of all jobs out of $\{j+1, \ldots, n\}$ that are not precedence-related to any of the jobs of W; denote this set by N. Then, if $r_W + r_N \leqslant R$, the subtree rooted at w can be discarded because each of the subsets of jobs in that subtree is resource-feasible. We also experimented with the exact method which decides if a given node of the tree leads to a forbidden set or not; see Problem (ii) of Section 4.2.2 for details. However, for the test sets we considered, the computational overhead was too large due to the time required to solve the associated minimum-flow problems.

Algorithm 5 shows further details of the proposed procedure. For a given node w of the tree, associated to some job j, and some job $i > j$, Algorithm 5 calls the subroutines *EvaluateNode(*i, w*)* and possibly also *CreateNode(*i, w*)*. The subroutine *EvaluateNode* computes the status of the set $W \cup \{i\}$, i. e., it decides

whether $W \cup \{i\}$ is (minimal) forbidden or resource-feasible. Algorithmic details are given in Algorithm 6. *CreateNode(i, w)* generates a new node of the tree which is a child of w and associated to job i.

Algorithm 5: Compute all minimal forbidden sets

Input : Jobs V, precedence constraints E_0, resource constraints r_j, R.

Output: The set of minimal forbidden sets represented by the tree $T(\mathcal{F})$.

Find suitable $k \in K$ and create ordering L of jobs with non-increasing r_{jk};
Compute Boolean matrix M which indicates whether $\exists F$ with $i, j \in F$;
$\mathcal{F} := \emptyset$; // stores the forbidden sets
$root :=$ root node of tree T; $Stack := \emptyset$;
for *all jobs* $j \in V$ **do**
    $w :=$ CreateNode($j, root$);
    push w on $Stack$;
while $Stack \neq \emptyset$ **do**
    remove node w from $Stack$;
    $j :=$ job associated to w; $W :=$ set of jobs associated to w;
    for *all jobs* $i >_L j$ **do**
        EvaluateNode(i, w);
        if $W \cup \{i\}$ *is a minimal forbidden set* **then**
            $u :=$ CreateNode(i, w);
            Add u to $\mathcal{F}$;
        if $W \cup \{i\}$ *is resource-feasible* **then**
            $u :=$ CreateNode(i, w);
            Add u to $Stack$;
    (Recursively) delete w if it is not minimal forbidden and has no children;
return $\mathcal{F}$;

4.3.5 Compact Representation of Forbidden Sets

Algorithm 5 immediately suggests to store all minimal forbidden sets in a data structure given by the tree $T(\mathcal{F})$. The jobs of the forbidden sets are represented as nodes in the tree, and upon building the tree as described before, a vector of pointers to the leaves of $T(\mathcal{F})$ is generated. To access (or loop over) all jobs of a forbidden set F, one simply traverses $T(\mathcal{F})$ from the leaf which corresponds to F to the root node, obviously in O($|F|$) time. In comparison to a representation as a vector of lists of the corresponding job numbers (which would certainly be

Algorithm 6: EvaluateNode (subroutine of Algorithm 5)

Input : A resource-feasible set W of jobs (represented by node w) and a new job i.

Output: Status of the set $W \cup \{i\}$ (resource-feasible / minimal forbidden / can be discarded).

if *i and some $j \in W$ cannot be in a minimal forbidden set w. r. t. M* **then**
  return *($W \cup \{i\}$ can be discarded);*

if *$r_{Wk} + r_{ik} > R_k$ for some k* **then**
  for $j \in W$ **do**
    if *$r_{Wk} + r_{ik} - r_{jk} > R_k$ for some k* **then**
      return *($W \cup \{i\}$ can be discarded);*
  return *($W \cup \{i\}$ is minimal forbidden);*

$N :=$ jobs of $\{j | V \ni j >_L i\}$ that potentially can be in some minimal forbidden set F with $(W \cup \{i\}) \subset F$ (according to Matrix M);

if $r_W + r_N \leqslant R$ **then return** *($W \cup \{i\}$ can be discarded);*
else return *($W \cup \{i\}$ is resource-feasible);*

the simplest data structure that provides fast access to traverse all minimal forbidden sets), this reduces memory requirement considerably (empirically analyzed in Section 4.4.2 below).

4.4 Computational Evaluation

We first describe the computational setup and the benchmark instances, and then analyze the performance of the proposed algorithm in dependence on different parameters which have been used to generate the instances.

4.4.1 Setup and Benchmark Instances

Our experiments were conducted on a Sun Ultra 1 with 143 MHz clock pulse operating under Solaris 2.7. The code is written in C++ and has been compiled with the GNU g++ compiler version 2.91.66 using the -O3 optimization option. The memory limit was set to 50 MB.

We have tested the proposed algorithm on instances of the library PSPLIB (2000) that was generated by Kolisch and Sprecher with the help of the instance generator ProGen (Kolisch and Sprecher 1996). The library contains instances

with 30, 60, 90, and 120 jobs, respectively. The instances have been generated by modifying three parameters, (i) the *network complexity* (NC) which is the average number of direct successors of a job, (ii) the *resource factor* (RF) which describes the average number of different resource types required in order to process a job divided by the total number of resource types, and (iii) the *resource strength* (RS), which is a measure of the scarcity of the resources (see (Kolisch and Sprecher 1996) for details). The parameters have been chosen out of the sets $NC \in \{1.5, 1.8, 2.1\}$ and $RF \in \{0.25, 0.5, 0.75, 1.0\}$. For the benchmark sets with 30, 60 and 90 jobs the resource strength RS has been chosen from the values $\{0.2, 0.5, 0.7, 1.0\}$, while for instances with 120 jobs it was chosen from $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. The smaller the parameter RS, the scarcer are the resources; hence, on average, the resource capacities are scarcer for the instances with 120 jobs. For each combination of the parameters, 10 instances have been generated at random. This results in 480 instances for each of instance sizes 30, 60, and 90, and 600 instances with 120 jobs. The number of resource types per instance is 4.

Before we turn to our computational experiences with these instances, let us briefly comment on the relationship between the systems of minimal forbidden sets and the above mentioned parameters RF and NC. According to Radermacher (1985, p. 237), instances are *essentially equal* if both precedence constraints and the systems of minimal forbidden sets coincide. In this respect, the variation of the resource factor RF does not necessarily lead to essentially different instances: Although two instances have a different resource factor, they can be identical in the sense that they have identical precedence constraints and systems of minimal forbidden sets. For example, if the threshold representation of the resource constraints defines a threshold hypergraph without isolated nodes (that is, one resource type suffices and for each $j \in V$ we have $r_{j,1} > 0$), the resource factor is obviously 1. However, the same system of minimal forbidden sets can be represented by $|\mathcal{F}|$ different resource types, which generally leads to a resource factor strictly smaller than 1. Another remark addresses the above definition of network complexity. Since the definition as the average number of direct successors disregards transitive precedence constraints, instances with identical network complexity may have an essentially different topology, hence also essentially different systems of minimal forbidden sets. For example, for $V = \{1, \ldots, 4\}$, the precedence constraints $E_0 := \{(1,2), (1,4), (3,4)\}$ and $E_0' := \{(1,2), (2,3), (3,4)\}$ both have a network complexity $NC = 3/4$. While (V, E_0') is a chain, and hence has no non-trivial anti-chain, (V, E_0) has three non-trivial anti-chains.

However, our computational results with the PSPLIB instances show that, on average, there is a meaningful correlation between all three parameters and the system of minimal forbidden sets.

4.4.2 Computational Results

Table 4.1 shows for each test set the number of solved instances (#solved), that is, all minimal forbidden sets could be computed within the memory restriction of 50 MB, as well as the average and maximum number of minimal forbidden sets ($\varnothing\ |\mathcal{F}|$ and max. $|\mathcal{F}|$) and required computation times ($\varnothing$ CPU and max. CPU). As the table suggests, the algorithm easily computes all minimal forbidden sets for the instances with 30 jobs; the computation time is negligible. Most of the instances with 60 jobs can also be solved in short time, however, there already exist few (17) instances for which not all minimal forbidden sets could be determined within the memory restriction of 50 MB (even with a limit 500 MB, 7 instances remain unsolved).

#jobs	#inst.	#solved	$\varnothing\ \lvert\mathcal{F}\rvert$	max. $\lvert\mathcal{F}\rvert$	$\varnothing$ CPU	max. CPU
30	480	480	326	4,411	0.01	0.2
60	480	463	101,773	2,163,692	7	167
90	480	309	255,476	1,867,239	23	490
120	600	340	243,871	1,996,505	13	200

Table 4.1: For each set of instances the table displays the number of instances in the test set (#inst.), the number of solved instances (#solved), the average and the maximum number of minimal forbidden sets ($\varnothing\ |\mathcal{F}|$ and max. $|\mathcal{F}|$), and the average and the maximum computation time in seconds ($\varnothing$ CPU and max. CPU).

Although for larger instances the average memory requirement strongly increases, the algorithm still solves more than a half of the instances with 90 and 120 jobs with no more than 50 MB memory requirement. Note that, even for instances with 120 jobs, for all instances with scarce resources ($RS = 0.1$) or small resource factor ($RF = 0.25$, that is, each job requires only one resource type on average), the algorithm computes all minimal forbidden sets at an average running time of less than 5 seconds. Instances with scarce resources are known to be particularly hard with respect to makespan minimization and lower bound computations.

Forbidden set statistics. Figures 4.2 and 4.3 show how the average number and cardinality of minimal forbidden sets depend on the instance parameters RS, RF, and NC. Since we did not observe that these parameters were significantly correlated, all figures are based on average values with respect to the whole set of instances (with 30 jobs). As expected, both the number and cardinality of minimal forbidden sets heavily depend on the instance parameters RS, RF, and NC; let us briefly analyze the outcome of this evaluation.

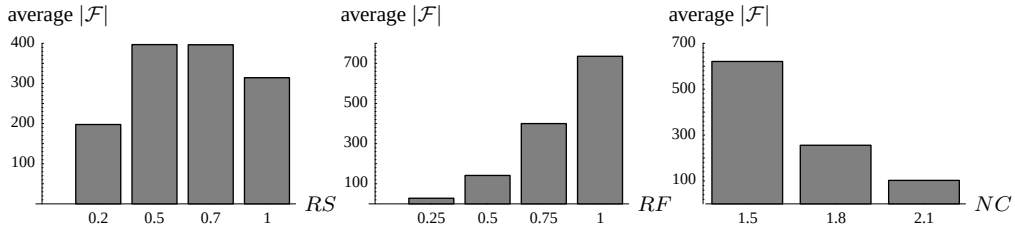

Figure 4.2: The plots display the average number of minimal forbidden sets depending on the instance parameters RS (left), RF (middle), and NC (right). The data is based on the test set with 30 jobs per instance.

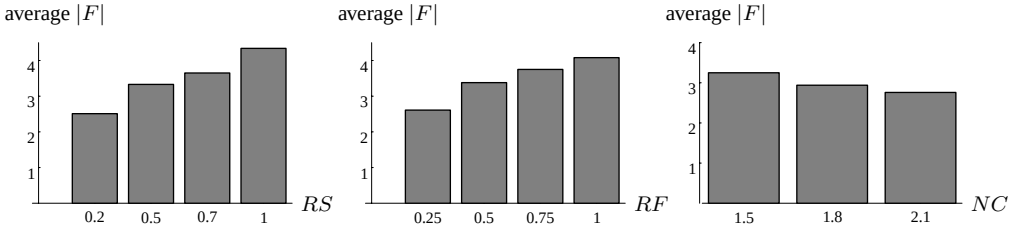

Figure 4.3: The plots display the average cardinality of minimal forbidden sets depending on the instance parameters RS (left), RF (middle), and NC (right). The data is based on the test set with 30 jobs per instance.

The dependence of the average cardinality of minimal forbidden sets on the resource strength RS as shown in Figure 4.3 is intuitive: If resources are scarce then the average cardinality is small and vice versa. With respect to the average number of minimal forbidden sets in dependence of the resource strength RS, it is noticeable that this figure is small either if the resource strength RS is very low (0.2; scarce resource capacity) or very high (1.0; loose resource capacity). For scarce resources, this is due to the fact that the minimal forbidden sets tend to be of small cardinality, as also suggested by Figure 4.3. For loose resources this is due to the fact that if there are hardly any resource constraints, already many anti-chains tend to be resource-feasible, hence there are fewer forbidden sets at all (with larger cardinality on average, as can be seen in Figure 4.3).

The behavior of the average cardinality of minimal forbidden sets in dependence of the resource factor RF in Figure 4.3 can be explained as follows. If each job requires only one or few resource types on average, that is, the resource factor RF is small, it is very likely that in a given anti-chain, there are pairs (i, j) of jobs with disjoint resource requirements ($r_{ik} \cdot r_{jk} = 0$ for all resource types k), hence minimal forbidden sets tend to be smaller on average the smaller the resource factor RF. (Consequently, there are also fewer of them, as can be seen in Figure 4.2.)

With respect to the network complexity NC, our results show that both number and cardinality of minimal forbidden sets trends down when NC increases.

The reason is that, for the considered instances, the total number of precedence constraints (including transitive ones) increases with the network complexity. Recall, however, that the network complexity is not a measure for the total number of precedence constraints in general (see Section 4.4.1).

We finally observed that the average cardinality of the minimal forbidden sets increases with the number of jobs. The respective average values (based on the number of solved instances as given in Table 4.1) are 3.5 (maximum 10) for 30 jobs, 4.9 (maximum 16) for 60 jobs, 5.1 (maximum 13) for 90 jobs, and 4.5 (maximum 12) for 120 jobs. Notice that the average and maximum cardinality is comparatively small for the test set with 120 jobs, which is due to the fact that the resource strength parameters are smaller for these instances (and perhaps also since quite some (260) of the 600 instances could not be solved within the 50 MB memory limitation).

Computational performance. Let us next analyze the computational performance with respect to running times, and compare the proposed algorithm (with and without reduction tests) to a variation of the earlier mentioned divide-and-conquer approach by Bartusch (1984); see Section 4.3. Table 4.2 first shows the average and maximal computation times for the algorithm proposed in this chapter, both with and without reduction tests.

	#jobs	#solved	∅ CPU	max. CPU
with reduction tests	30	480	0.01	0.2
no reduction tests	30	480	0.04	0.5
with reduction tests	60	463	7	167
no reduction tests	60	446	145	6,280

Table 4.2: For both variations of the algorithm, the table displays the number of solved instances (#solved) and the respective average and the maximum computation time in seconds (∅ CPU and max. CPU); based on instances with 30 and 60 jobs, respectively.

The results obviously confirm that the additional reduction tests proposed in Section 4.3.4 are worthwhile, reducing the average required computation time by a factor of almost 4 for the instances with 30 jobs. The importance of the additional reduction tests is even more apparent for the instances with 60 jobs. There, the average computation time decreases from 145 seconds (without reduction tests) to 7 seconds (with reduction tests); a factor of more than 20.

Figure 4.4 shows more details with respect to the computation times, based on the test set with 30 jobs. It is intuitive that the computation times are small

whenever there are only few, and small minimal forbidden sets, and large if there are many and large minimal forbidden sets. This is indeed validated by Figure 4.4. There is however, one more remark on the computation times which concerns the reduction tests proposed in Section 4.3.4 (see also Table 4.2): If the reduction tests are not performed, the dependence of computation time on the resource factor RF gives a picture which is exactly reverse, showing that these tests are extremely effective particularly for instances where the resource factor RF is small. In fact, for $RF = 0.25$, the computation time increases from 1.4 ms (Figure 4.4; with reduction tests) to 41 ms (without reduction tests).

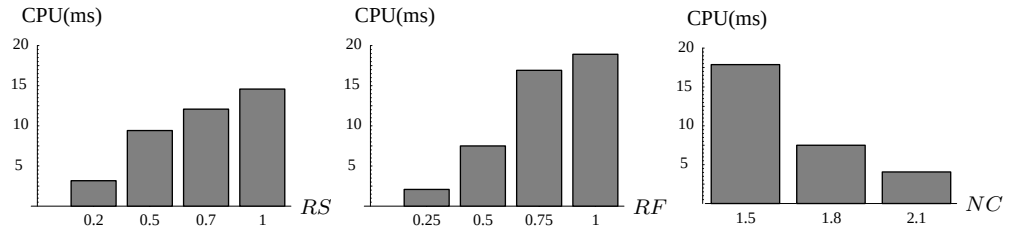

Figure 4.4: The plots display the average running time (in milliseconds) depending on the instance parameters RS (left), RF (middle), and NC (right). The data is based on the test set with 30 jobs per instance.

We have also experimented with a divide-and-conquer approach, based on a previously proposed approach by Bartusch (1984). The basic idea is to partition the given instance, say I, into $|K|$ partial instances $I_1, \ldots, I_{|K|}$ where each I_k only consists of jobs which require a positive amount of resource k. Then, for each I_k, the set of minimal forbidden sets $\mathcal{F}_k$ is calculated with respect to resource k only. This has the major advantage that each of the subproblems can be solved in polynomial time with respect to its in- and output, which is n and $|\mathcal{F}_k|$, respectively (see Theorem 4.2.2 and Section 4.3.3). On the other hand, it is obvious that the systems of minimal forbidden sets $\mathcal{F}_k$ for the subproblems I_k may be exponential with respect to $\mathcal{F}$ itself, and the efficient computation of the inclusion-minimal subsets of $\bigcup_k \mathcal{F}_k$ constitutes a non-trivial problem in its own.

Based on the instances from the PSPLIB, we have compared the time required to compute $\mathcal{F}$ using the algorithm proposed in this chapter with the overall time required time to compute all minimal forbidden sets $\mathcal{F}_k$ for all partial instances I_k, $k = 1, \ldots, |K|$. It turned out that these computation times are in fact comparable on average, however, for only few instances the divide-and-conquer approach was more efficient. In particular, notice that this comparison does not even take into account the additional overhead required to compute the inclusion-minimal subsets of $\bigcup_k \mathcal{F}_k$. In fact, using a straightforward implementation, this overhead turned out to be a major bottleneck of the divide-and-conquer approach; it required by far more computation time than the computation of the minimal forbidden sets

$\bigcup_k \mathcal{F}_k$ itself. Hence, a divide-and-conquer approach seems to be beneficial only for instances with very particular structure (e. g. Example 4.5.2 in Section 4.5).

Memory requirements. Finally, we analyze the memory required to store the minimal forbidden sets in the data structure given by the tree $T(\mathcal{F})$, in comparison to the ordinary list representation, where each minimal forbidden set is stored as a list of job numbers. For the instances with 30 jobs, the average sum $\sum_{F\in\mathcal{F}} |F|$ is roughly 1400, while the average number of nodes in T is only 600. Despite of the fact that we have to maintain some overhead in order to generate (and delete) the tree $T(\mathcal{F})$, namely an integer which counts the number of children of each node in the tree, the memory requirement is reduced by a factor of roughly 1.5 in comparison to the list representation. For instances with 60, 90, and 120 jobs, the memory requirements are reduced by a factor of roughly 2.5. (This value only refers to instances for which all minimal forbidden sets could be computed within the given memory limitation of 50 MB.) Consequently, compared to the ordinary list representation, the proposed data structure given by $T(\mathcal{F})$ allows a much more efficient representation of minimal forbidden sets.

4.5 Further Remarks and Examples

There are some open questions which remain for future research: The question if some given job i is contained in some minimal forbidden set or not (given a threshold representation of the resource constraints) was proved to be NP-complete, even for the case without precedence constraints using a reduction from PARTITION (Theorem 4.2.3). In fact, it is not hard to see that this problem can be solved in pseudo-polynomial time by iteratively solving SUBSETSUM problems. However, it is open whether the problem becomes strongly NP-hard if also precedence constraints are present. Another interesting open problem is the question whether minimal forbidden sets can be computed in time polynomial in the in- and the output size of the problem if the number of resource types is greater than one, which would generalize Theorem 4.2.2.

We finally present two examples that have been referenced in the text. Example 4.5.1 exposes that the number of resource types that are required to model a system of minimal forbidden sets may be exponential in n, the number of jobs (see Section 4.2.1). Example 4.5.2 shows that the number of nodes in the search tree that are produced with Algorithm 5 may be exponential in $|\mathcal{F}|$ (see Section 4.3.3).

Example 4.5.1. *Let $V = \{1, \ldots, 4n\}$, $E_0 = \emptyset$, and let $V_1 = \{1, \ldots, 2n\}$ and $V_2 = \{2n+1, \ldots, 4n\}$ be a partition of V. Now, for each $U_1 \subseteq V_1$ define a corresponding $U_2 := \{u + 2n \mid u \in U_1\}$, and let $\mathcal{F} := \{U_1 \cup U_2 \mid U_1 \subset V_1, |U_1| = n\}$ be the minimal forbidden sets.*

Here, the number of minimal forbidden sets is $\binom{2n}{n} \in \Omega(\, 2^n \,)$. Now, for any two distinct minimal forbidden sets, say $U_1 \cup U_2$ and $W_1 \cup W_2$, where $U_1, W_1 \in V_1$ and $U_2, W_2 \in V_2$ according to the above definition, two different resource types are required. Otherwise at least one of the sets $U_1 \cup W_1$, $U_2 \cup W_2$, $U_1 \cup W_2$, or $U_2 \cup W_1$ would not be resource-feasible. Hence, $\Omega(\, 2^n \,)$ resource types are required to represent $\mathcal{F}$. In other words, the threshold dimension of $(V, \mathcal{F})$ is $\Omega(\, 2^n \,)$. □

Example 4.5.2. *Let* $V = \{1, \ldots, n\}$*,* $E_0 = \emptyset$*,* $|K| = 2$*,* $R_1 = R_2 = 2n - 4$*,* $r_{1,1} = r_{2,1} = r_{n-1,2} = r_{n,2} = n$*, and* $r_{jk} = 1$ *otherwise.*

Then $\mathcal{F}$ consists of exactly six sets, namely $\{1, 2\}$, $\{n-1, n\}$, and $\{i, 3, 4, \ldots, n-3, n-2, j\}$ for $i = 1, 2$ and $j = n-1, n$. Hence $|\mathcal{F}| \in \mathrm{O}(\, 1 \,)$ for any $n \in \mathbb{N}$, but the number of nodes which are examined within our algorithm is exponential in n. Notice that this is an example where the divide-and-conquer approach by Bartusch (see Section 4.3) is beneficial, since it runs polynomial in n. □

CHAPTER 5

ROBUST SCHEDULING POLICIES

At a first glance, the material that we have presented in the Chapters 2–4 shows no direct relationship to the main issue of this thesis, the stochastic resource-constrained project scheduling problem. In this chapter, we discover the connecting link between stochastic resource-constrained project scheduling and the previous results on AND/OR precedence constraints. The link is a particular class of scheduling policies, the class of *preselective policies*. Based on the observation that a preselective policy can be expressed as a set of AND/OR precedence constraints, we present results on domination of policies as well as on the computation of earliest job start times. We also study different subclasses of the class of preselective policies. In fact, the results presented in this chapter are the key to a successful computation of 'optimal' policies for stochastic resource-constrained projects; we computationally apply the results in Chapter 6 below.

Most of the material that is presented in this chapter has been published in (Möhring and Stork 2000).

5.1 Introduction

We consider the stochastic resource-constrained project scheduling problem as defined in Chapter 1. Recall from the introduction and Chapter 1 that, due to the combination of random job processing times and resource constraints, a project is executed according to a so-called *policy* (sometimes also called *strategy*). A scheduling policy (or *policy*, for short) may be seen as a dynamic decision process that defines which jobs are started at certain decision times t, based on the knowledge of the observed past up to t. The best-known class of such scheduling policies is certainly the class of *priority policies*. A policy is called a *priority policy* if at any time t a maximal number of available jobs is scheduled according to a given priority order on the set of jobs. Here, a job is called *available* at time t if it is not yet started and all its predecessors have already been completed by time t. While priority policies are easy to define and implement, they have several well known drawbacks. For example, there are instances (even with deterministic processing times) where no priority policy yields an optimal schedule. Moreover, a change in the job processing times may lead to anomalies such as an increase

of the project duration although job processing times have been decreased, see (Graham 1966); we also give an example in Section 5.2 below. Such effects are sometimes called *Graham anomalies*. As a consequence of the Graham anomalies, we think that priority policies are not the first choice to execute stochastic resource-constrained projects. Instead, our work is based on so-called *preselective policies* that have been introduced by Radermacher (1981b) and were later studied by Igelmund and Radermacher (1983b, 1983a). Such policies define for each minimal forbidden set a *preselected* job $j \in F$ which is postponed until at least one job from $F \setminus \{j\}$ has been completed. In contrast to priority policies, preselective policies do not show the undesired Graham anomalies. In this sense, they are more 'robust' compared to priority policies.

In this chapter we show that preselective policies can be expressed as a set of AND/OR precedence constraints. This interpretation yields new insights into the combinatorial structure of preselective policies and allows to derive a necessary and sufficient dominance criterion. In addition, we employ the results of Chapter 3 to derive an efficient, scenario-based algorithm that (approximately) computes the expected cost when a project is executed according to a specific preselective policy. Next, to further simplify the computational treatment, we introduce a new subclass of the class of preselective policies, the so-called *linear preselective policies*. They combine the simplicity of priority policies with the structural attractiveness of preselective policies by defining the preselective jobs according to *priority lists*. Linear preselective policies inherit all the favorable properties of preselective policies but are considerably better tractable from a computational point of view. Unfortunately, like preselective policies, linear preselective policies also require the representation of resource-constraints by (possibly exponentially many) minimal forbidden sets. We therefore study another subclass of preselective policies, we call them *job-based priority policies*. They do not require the forbidden set representation of resource constraints; the usual threshold representation suffices. As a consequence, algorithms that are based on job-based priority policies have the potential to be applicable to projects where a large number of minimal forbidden sets makes the use of forbidden set based policies computational inefficient. The computational simplifications, however, are not without pay: The optimum (expected cost) values that can be achieved within the classes of linear preselective policies and job-based priority policies are generally worse in comparison to preselective policies. However, the gap usually seems to be rather small, as our computational results presented in Chapter 6 below expose.

The chapter is organized as follows. In the next section we briefly review previous work on scheduling policies and discuss the class of priority policies. Next, in Section 5.3, we define the class of *Earliest start policies*. For this class, we do not establish new insights, however, we report on a branch-and-bound algorithm based on ES-policies in Chapter 6 below. Sections 5.4–5.6 are concerned with a

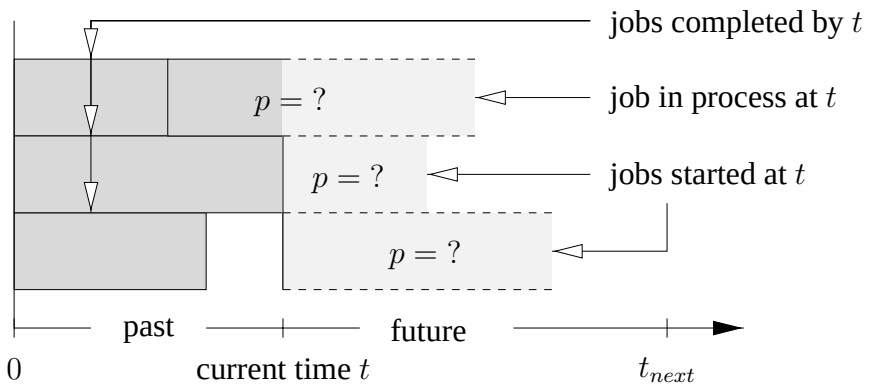

Figure 5.1: The picture shows the action that is performed at the current time t. Two jobs are started, and a new tentative decision time t_{next} is defined. At time t, three jobs have already been completed and one job was started before t and is still in process.

discussion of the classes of preselective policies, linear preselective policies, and job-based priority policies. In Section 5.7 we relate the classes of policies to each other with respect to the optimum value that can be achieved within the respective class.

5.2 General Scheduling Policies

In the context of resource-constrained project scheduling, policies have been introduced by Radermacher (1981b). He adapts the theory of *stochastic dynamic programming* to the specific scheduling application and studies various classes of policies and their properties. Generally, stochastic dynamic programming is concerned with random processes that change their state over time. To control a process, actions may be chosen at some time t; and based on the state of the process at t and the action chosen, the probability distribution of the next state is determined. Actions have to be chosen such that some given goal is met or some cost function is optimized. Let us refer to the work of Kaerkes, Möhring, Oberschelp, Radermacher, and Richter (1981) who discuss additional details how Radermacher's concept of a scheduling policy is embedded into the theory of stochastic dynamic programming. There exist several textbooks that give an introduction to stochastic dynamic programming, we refer to (Ross 1983).

There are several alternative views on scheduling policies. The view of a policy as a dynamic decision process is probably most intuitive. Independent from Radermacher's work, this viewpoint has been taken by Fernandez, Arma-

cost, and Pet-Edwards (1998a, 1998b) to establish a definition of policies for stochastic resource-constrained project scheduling. A policy defines *actions* at *decision times*. In our case an action consists of a set $B_t \subseteq V$ of jobs to be started at the current decision time t and the definition of a tentative next decision time $t_{next} > t$. The situation at some decision time t is illustrated in Figure 5.1. Once the jobs from B_t are started, the processing of the jobs is observed over time until either t_{next} is reached or some job completes. The event which occurs earlier defines the next decision time. Throughout the text, we always assume that the first decision time is $t = 0$, i. e., the project starts at time 0. Clearly, the set B_t must be defined appropriately. This is, on the one hand, the requirement that B_t contains no job that has been started at a previous decision time, and no precedence or resource constraint is violated once the jobs of B_t are started at t. On the other hand, to define the set B_t, a policy may only use information that is available at time t. This information includes the input data (set of jobs, precedence constraints, resource constraints, processing time distributions) and the conditional distributions of job processing times that result from $\boldsymbol{p}$ and the observed past up to time t. This requirement is often called *non-anticipativity constraint* and appears in various publications on stochastic optimization problems, mostly in conjunction with the analysis of a set of scenarios. For more details we refer to (Wets 1989; Rockafellar and Wets 1991; Escudero, Kamesam, King, and Wets 1993; Mulvey, Vanderbei, and Zenios 1995; Birge and Dempster 1996). In the context of resource-constrained project scheduling, Fernandez and Armacost (1996) note that various commercial software packages violate this important requirement and hence produce misleading results.

Once every job has been completed, the above described dynamic decision process stops. At that time we know the processing times of each job and thus have a scenario p of job processing times. In fact, every policy Π may alternatively be interpreted as a function $\Pi : \mathbb{R}^n_> \to \mathbb{R}^n_\geqslant$ that maps given scenarios p of job processing times to vectors $S(p) \in \mathbb{R}^n_\geqslant$ of feasible job start times (*schedules*). We denote the start time of a job $j \in V$ for a given policy Π and a given scenario p by $S^\Pi_j(p)$ and its completion time by $C^\Pi_j(p) := S^\Pi_j(p) + p_j$ (recall Chapter 1). If no misinterpretation is possible we omit the policy superscript Π.

The discussion so far suggests that there are two different views on a policy, namely the view as a *dynamic decision process* and a *function*. In fact, there is another view which is particularly important for computational issues. A policy can often be represented by a pair which consists of a combinatorial object (e. g., an ordering of the jobs) and an algorithm which transforms, for a given scenario p of processing times, the combinatorial object into a schedule $S(p)$. Hence, the pair implicitly defines for each possible scenario and each decision point t the set B_t of jobs to be started. Each of the three views on policies is useful for different purposes. Radermacher (1981b) used the view as a function to formally define

the term *policy*, which was later used in many other publications (co-authored) by Radermacher, we refer to (Radermacher 1981a; Igelmund and Radermacher 1983b; Igelmund and Radermacher 1983a; Möhring, Radermacher, and Weiss 1984; Möhring, Radermacher, and Weiss 1985; Möhring and Radermacher 1985; Radermacher 1986) for a comprehensive characterization of policies and the discussion of special classes of policies.

Definition 5.2.1. *(Radermacher 1981a) A function* $\Pi : \mathbb{R}^n_> \to \mathbb{R}^n_\geqslant$, $p \xrightarrow{\Pi} S^\Pi(p)$, *is called a* policy *if the following four conditions hold.*

(i) $C^\Pi_i(p) \leqslant S^\Pi_j(p)$ *for all* $(i,j) \in E_0$ *and* $p \in \mathbb{R}^n_>$ $(C^\Pi_i(p) := S^\Pi_i(p) + p_i)$,

(ii) $F \not\subseteq \{j \in V | S^\Pi_j(p) \leqslant t < C^\Pi_j(p)\}$ *for all* $t \in \mathbb{R}_\geqslant$, $F \in \mathcal{F}$, *and* $p \in \mathbb{R}^n_>$,

(iii) If $S^\Pi_j(p) = t$ *for some arbitrary* $j \in V$, $p \in \mathbb{R}^n_>$, *and* $t \in \mathbb{R}_\geqslant$, *then* $S^\Pi_j(p') = t$ *for all* $p' \in \mathbb{R}^n_>$ *with the two following properties: if* $C^\Pi_i(p) \leqslant t$ *then* $p_i = p'_i$ *and if* $S^\Pi_i(p) \leqslant t < C^\Pi_i(p)$ *then* $p'_i > t - S^\Pi_i(p)$,

(iv) Π *is universally measurable and avoids total idle times (a total idle time is a time period where no job is in process but the project has not been completed at the beginning of that time period).*

Let us briefly explain the definition. Properties (i) and (ii) simply mean that for each scenario p of job processing times S^Π is a feasible schedule with respect to G_0, $\mathcal{F}$, and p: Property (i) ensures that all precedence constraints are respected, while Property (ii) avoids the simultaneous processing of a forbidden set. The requirement of *non-anticipativity* is formulated in Property (iii), i. e., the decision at any time t is only based on information that is available by time t. In fact, Radermacher (1981b, Theorem 1.5) has shown that Property (iii) exactly expresses the possibility of using the maximal amount of available information for each t (see also (Möhring, Radermacher, and Weiss 1984, Theorem A)). Finally, Property (iv) is necessary to guarantee the existence of the project cost distribution (associated with Π) and ensures that the expected project cost of a policy is finite. Kaerkes, Möhring, Oberschelp, Radermacher, and Richter (1981, Example 12.4) gave an example which demonstrates that $\kappa(C^\Pi(p))$ as a function of p (where Π fulfills Properties (i)–(iii)) needs not necessarily be universally measurable. For each class of policies that is considered in the Sections 5.3– 5.6 below, Property (iv) is redundant: it follows from Properties (i)–(iii) and the definition of the respective policy. For a more detailed discussion of Definition 5.2.1 we refer to (Radermacher 1981b; Möhring, Radermacher, and Weiss 1984; Möhring and Radermacher 1985).

Next, we define what is meant by an *optimal* policy.

Definition 5.2.2. *Let τ be a class of policies. We define a policy Π^* to be optimal with respect to τ if Π^* minimizes the expected project costs within τ, i. e., $E[\kappa(C^{\Pi^*}(\boldsymbol{p}))] = \inf\{E[\kappa(C^{\Pi}(\boldsymbol{p}))] : \Pi \in \tau\}$. We denote the cost of an optimum policy of class τ by ρ^τ .*

The generality of Definition 5.2.1 suggests that, for the model of resource-constrained project scheduling, there is no hope that the class of all *all* policies is computationally tractable. One therefore usually restricts to subclasses which have a simple combinatorial representation and where decision points are restricted to be $t = 0$ (project start) and job completions, only. Policies with the latter property are sometimes called *elementary*, in fact, we only consider elementary policies. However, tentative decision points are sometimes useful, e. g., in (Skutella and Uetz 2001) where an approximation algorithm for precedence-constrained parallel machine scheduling is derived which is based on a non-elementary class of policies. This is the only currently known approximation algorithm with constant performance guarantee for that model.

As an example for a class of elementary policies, let us consider the class of *priority policies* which is certainly the best-known class of policies. There is a considerable amount of literature in which priority policies are analyzed for particular machine scheduling models. A priority policy Π is characterized by an ordering L of the jobs. For each decision time t ($t = 0$ or a job completion), all jobs j not yet started are considered in the order of L. Π schedules j at time t if this does neither violate a precedence constraint nor a resource constraint. In deterministic scheduling this is well-known as *Graham's List Scheduling* (Graham 1966), it is sometimes also called the *parallel list scheduling scheme*, e. g., in (Kolisch 1996). However, due to their greedy usage of resources, priority policies have several drawbacks when applied to precedence-constrained models. In particular, there exist instances (even with deterministic processing times) in which no priority policy yields an optimal schedule. Moreover, the change of job processing times may lead to so-called Graham anomalies such as an increase of the project duration although job processing times have been decreased, see (Graham 1966). Thus, if we think of a policy as a function that maps a scenario of job processing times to feasible start times, priority policies are neither monotone nor continuous. The following example, which is taken from (Möhring 2000b) demonstrates this undesired property.

Example 5.2.3. *Let $G_0 = (V, E_0)$ be given by $V = \{1, \ldots, 7\}$ and $E_0 = \{(1, 4), (1, 5), (2, 3), (3, 4), (3, 5), (3, 7), (4, 6), (5, 6)\}$ and let the set $\{4, 5, 7\}$ be minimal forbidden. Moreover, job processing times are given by $p = (4, 2, 2, 5, 5, 10, 10)$.*

For the ordering $L = 1 \prec \cdots \prec 7$ Graham's list scheduling yields a makespan of 19. If each processing time is decreased by 1, the same ordering L yields a

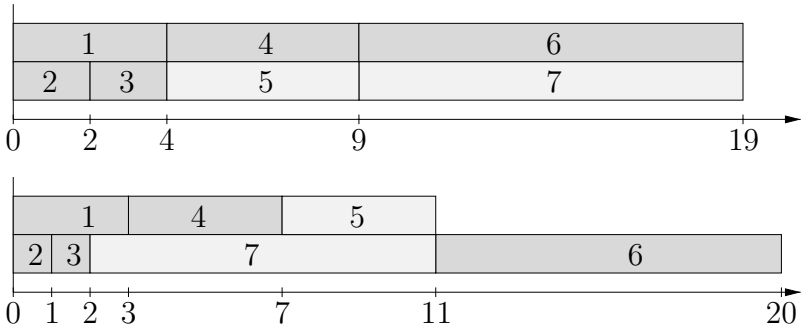

Figure 5.2: The figure shows two schedules that result from Example 5.2.3 and the scenarios $p = (4, 2, 2, 5, 5, 10, 10)$ and $p' = (3, 1, 1, 4, 4, 9, 9)$, respectively.

makespan of 20. The start time of jobs 5 and 7 jump discontinuously if p changes continuously to $p' = (3, 1, 1, 4, 4, 9, 9)$. The schedules which result from the two scenarios are depicted in Figure 5.2. Note that with respect to p' no priority policy as defined above yields an optimal schedule.

As a consequence of the above discussion, priority policies are considered inadequate for planning stochastic resource-constrained projects. Extending a definition in (Möhring 2000b), a minimal requirement for a *robust* policy is monotonicity and continuity (in view of policies as functions). Hence, in the sequel, we are going to focus on ES-policies, (linear) preselective policies, and job-based priority policies. Each such policy is monotone and continuous and consequently, Graham anomalies do not occur within project execution.

Finally, recall that we sometimes make use of dummy jobs a and b which mark the project start and the project completion, respectively, i. e., there are precedence constraints (a, j) and (j, b) for each $j \in V$. The dummy jobs have fixed processing time 0 and consume no resources.

5.3 Earliest Start Policies

The first robust class we consider are so-called *Earliest Start policies*, or *ES-policies*, for short. Within the context of stochastic resource-constrained project scheduling, the class was introduced by Radermacher (1981b) and further studied in (Radermacher 1981a). We do not establish new theoretical insights for ES-policies, however, they are of interest for the following reasons. First, the class of ES-policies is closely related to the class of preselective policies; it is in fact a subclass thereof. Second, we included ES-policies into our computational considerations that are presented in Chapter 6 below. The simple and intuitive idea is to extend the given partially ordered set $G_0 = (V, E_0)$ to a partially ordered set

$G = (V, E)$, $E_0 \subseteq E$, such that no minimal forbidden set is an anti-chain in G. Then, in order to obtain a feasible schedule $S(p)$ for a given scenario p of job processing times, an ES-policy simply computes earliest job start times with respect to G, i. e., $S_a(p) := 0$ and

$$S_j(p) := \max_{(i,j)\in E}(S_i(p) + p_i) \qquad j \in V \ . \tag{5.1}$$

This concept is quite common in deterministic scheduling theory, e. g., within the disjunctive graph model for shop-scheduling problems (where each minimal forbidden set is of cardinality 2).

The above description of an ES-policy follows the view of a policy as a pair of a combinatorial object and an algorithm. Here, the combinatorial object is the extension G of G_0 in which no minimal forbidden set is an anti-chain. The algorithm to transform a given scenario $p \in \mathbb{R}^n_>$ to a schedule is a classical longest path algorithm defined by (5.1). With this discussion we obtain the following definition of an ES-policy.

Definition 5.3.1. *(Radermacher 1981b) A policy* Π *(as in Definition 5.2.1) is called* ES-policy *if there is a partially ordered set* $G = (V, E)$ *such that, for all* $j \in V$ *and* $p \in \mathbb{R}^n_>$, $S^\Pi_j(p)$ *equals the start times obtained from* (5.1).

To give an example, consider the ES-policy for Example 1.3.1 defined by $E = E_0 \cup \{(1,5), (2,3), (4,5)\}$. Obviously, non of the minimal forbidden sets $\{1,5\}$, $\{2,3,4\}$ and $\{2,4,5\}$ of Example 1.3.1 is an anti-chain in E. For the scenario $p = E[\boldsymbol{p}]$ the recursive Formula (5.1) yields start times $S(p) = (0,0,5,3,8)$.

Definition 5.3.1 immediately suggests that ES-policies are elementary. Moreover, notice that job start times are defined by a composition of *sums* and *maxima* of processing times. Consequently, if ES-policies are viewed as functions $\Pi : p \to S(p)$, the definition implies that ES-policies are monotone, continuous, and convex. In fact, Radermacher (1981b, Theorem 2.19) also established the following, opposite result: Any convex policy is an ES-policy (see also (Radermacher 1986)).

Let us next discuss *domination* properties of ES-policies. Generally, we call a policy Π *dominated* if there exists another policy $\Pi' \neq \Pi$ such that $S^\Pi_j(p) \geqslant S^{\Pi'}_j(p)$ for all scenarios $p \in \mathbb{R}^n_>$ and all jobs $j \in V$. We here assume that Π and Π' belong to the same class of policies. For ES-policies, the following lemma is easy to show.

Lemma 5.3.2. *(Radermacher 1981b) Let the partially ordered sets* (V, E) *and* (V, E') *be extensions of* (V, E_0) *that represent ES-polices* Π *and* Π', $\Pi \neq \Pi'$, *respectively.* Π *is dominated by* Π' *if and only if* $E' \subseteq E$.

A proof of this observation is given in Section 5.4.2 below where a generalization of the lemma is discussed (see Lemma 5.4.4). In Chapter 6 we see that the concept of domination is in fact a key to effective branch-and-bound algorithms.

5.4 Preselective Policies

The class of *preselective policies* was introduced by Radermacher (1981b) and further studied by Igelmund and Radermacher (1983b, 1983a). They give two different combinatorial representations of preselective policies and develop a branch-and-bound algorithm which computes an optimal policy in the class. In contrast to ES-policies, preselective policies choose for each minimal forbidden set F one job $j \in F$ and never execute j before at least one $i \in F \setminus \{j\}$ has been completed. Under the name *delaying alternative*, a related approach of solving resource conflicts became later a very popular tool in deterministic resource-constrained project scheduling (see, e. g., (Demeulemeester and Herroelen 1992; Schwindt 1998; Fest, Möhring, Stork, and Uetz 1998)).

5.4.1 Definition and Characteristics

Let us start by describing one of the combinatorial representations of a preselective policy that have been suggested by Igelmund and Radermacher (1983b). For a system $\mathcal{F} = \{F_1, \ldots, F_f\}$ of minimal forbidden sets, a *selection* is a sequence $s = (s_1, \ldots, s_f)$ such that $s_\ell \in F_\ell$ for all $\ell \in \{1, \ldots, f\}$. The intended meaning is that the start of s_ℓ is postponed until at least one job $i \in F_\ell \setminus \{s_\ell\}$ has been completed, within other words $S_{s_\ell}(p) \geqslant \min_{i \in F_\ell \setminus \{s_\ell\}}(S_i(p) + p_i)$ is valid for all scenarios $p \in \mathbb{R}^n_>$. We call the jobs s_ℓ *preselected* jobs.

Next, in order to define a preselective policy, we need an algorithm that describes how a schedule $S(p)$ is computed from a given selection s and a given scenario $p \in \mathbb{R}^n_>$. However, for the moment, assume that we start each job as early as possible subject to the constraints given by E_0 and the selection s. We will see below that this is in fact well defined. Before, let us consider an alternative characterization of a preselective policy that was studied by Igelmund and Radermacher (1983b). A preselective policy with selection s can be represented by a collection of partially ordered sets $G = (V, E)$ each of which extends the given partial order G_0 of precedence constraints and *respects* the selection s. That is, $E_0 \subseteq E$ and for each minimal forbidden set F_ℓ with preselected job $j_\ell \in F_\ell$, E contains an ordered pair (i, j_ℓ) with $i \in F_\ell \setminus \{j_\ell\}$. Consider Example 1.3.1 where the minimal forbidden sets are ordered as $F_1 = \{1, 5\}$, $F_2 = \{2, 3, 4\}$, and $F_3 = \{2, 4, 5\}$ (we keep this ordering for the rest of the thesis). For selection $s = (5, 4, 2)$ the construction of a set of extensions of G_0 that *respect* s is depicted

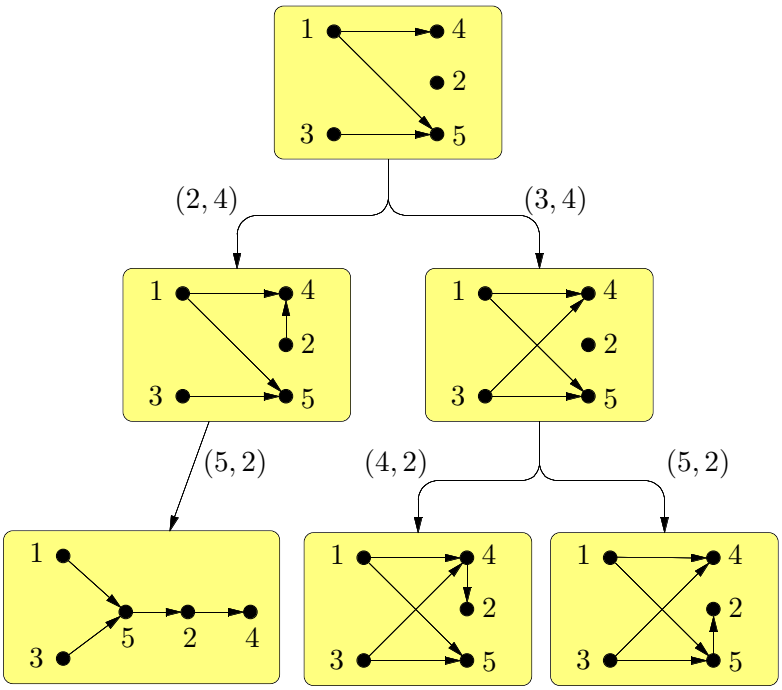

Figure 5.3: The leaves of the (rooted) tree represent three possible extensions of G_0 that respect the selection $s = (5, 4, 2)$. The pairs on the arcs denote the precedence constraints that are added to E_0.

in Figure 5.3.

Now, notice that this characterization is closely related to the definition of a *realization* for a set of waiting conditions; recall the discussion in Section 2.2. In fact, the concept of AND/OR precedence constraints suggests another, very useful representation of preselective policies. Each restriction induced by a minimal forbidden set F and its preselected job j can be represented by the waiting condition $(F \setminus \{j\}, j)$. Moreover, each given precedence constraint $(i, j) \in E_0$ can obviously be represented by the waiting condition $(\{i\}, j)$. Thus, instead of considering precedence constraints, minimal forbidden sets and a selection, it suffices to consider a set $\mathcal{W}$ of waiting conditions. We say that the so-defined set $\mathcal{W}$ of waiting conditions is *induced* by s and E_0. Let us carry over the notion of a realization from Chapter 2: We call a partially ordered set $G = (V, E)$ a *realization* of $G = (V, E)$ and s if G extends G_0 and respects the selection s.

Recall from Chapter 2 that a set of waiting conditions induces a digraph D which has a node for each job and for each waiting condition. There is a directed arc from a node representing a job i to a node representing a waiting condition (X, j) if $i \in X$. Furthermore, each node representing a waiting condition (X, j)

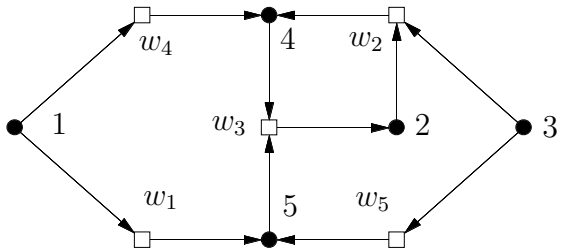

Figure 5.4: The digraph resulting from Example 1.3.1 and selection $s = (5, 4, 2)$. Circular nodes correspond to jobs while square nodes represent waiting conditions. Nodes w_1, w_2, and w_3 are induced by minimal forbidden sets and the respective preselected job. Precedence constraints are represented by w_4 and w_5.

is connected to the node representing j.

The set of waiting conditions which is induced by Example 1.3.1 with selection $s = (5, 4, 2)$ is $\mathcal{W} = \{w_1 = (\{1\}, 5), w_2 = (\{2, 3\}, 4), w_3 = (\{4, 5\}, 2), w_4 = (\{1\}, 4), w_5 = (\{3\}, 5)\}$. The associated digraph D which represents $\mathcal{W}$ is depicted in Figure 5.4. Although D contains cycles, there is a possible ordering in which jobs can be executed (e. g., $1 \prec 3 \prec 4 \prec 5 \prec 2$). However, like with sets of waiting conditions, selections may be *infeasible*. This is the case if and only if there exists a generalized cycle in the associated digraph D of waiting conditions (see Lemma 2.3.2 and also Igelmund and Radermacher (1983a, Theorem 1.1)). By Theorem 2.3.1 and Algorithm 1, the question if a given selection is feasible or not can be decided in linear time (in the encoding of $\mathcal{W}$). Consider Example 1.3.1 with selection $s = (1, 3, 2)$. The associated digraph of waiting conditions is a generalized cycle, hence, s is infeasible.

For a given feasible selection s and resulting system of waiting conditions $\mathcal{W}$ we can construct a feasible schedule for each possible scenario $p \in \mathbb{R}^n_>$ by setting $S_a(p) := 0$ and

$$S_j(p) := \max_{(X,j)\in\mathcal{W}} (\min_{i\in X}(S_i(p) + p_i)) \qquad j \in V \ . \tag{5.2}$$

Recall from the discussion in Section 3.1 that (5.2) defines unique, componentwise minimal job start times. To compute start times according to (5.2), Igelmund and Radermacher (1983a) propose an algorithm with a running time $\mathrm{O}(mn^2)$, where, m is the number of arcs in the digraph which represents $\mathcal{W}$. However, the insight to represent a preselective policy by a set of waiting conditions suggests a more efficient alternative: We can apply Algorithm 3, which results into a running time of $\mathrm{O}(n + m + f \log f)$ where $f = |\mathcal{F}|$. Note that, since $f \in \mathrm{O}(\, 2^n\,)$, the algorithm has a lower worst case running time than the one proposed by Igelmund and Radermacher (1983a). Let us re-consider Example 1.3.1. For

the selection $s = (5, 4, 2)$ and the scenario $p = E[\boldsymbol{p}]$, Formula (5.2) yields $S(E[\boldsymbol{p}]) = (0, 8, 0, 3, 3)$.

With the above formulation of how a given scenario p is transformed into a schedule, we obtain the following notion of a preselective policy.

Definition 5.4.1. *A policy* Π *(as in Definition 5.2.1) is called* preselective policy *if there is a set* $\mathcal{W}$ *of waiting conditions such that, for all* $j \in V$ *and* $p \in \mathbb{R}^n_>$, $S^\Pi_j(p)$ *equals the start times obtained from* (5.2)*. Moreover,* Π *is preselective* with selection s *if* $\mathcal{W}$ *is induced by* s *(and* E_0*).*

Notice that the definition differs slightly from the original definition of a preselective policy of Radermacher (1981b, Section II, 2.), see also (Igelmund and Radermacher 1983b, Section IV). Radermacher defined a policy to be preselective with selection $s = (s_1, \ldots, s_f)$, if $S_{s_\ell}(p) \geqslant \min_{i \in F_\ell \setminus \{s_\ell\}}(S_i(p) + p_i)$ is valid for all scenarios $p \in \mathbb{R}^n_>$ and all $\ell \in \{1, \ldots, f\}$. He did not specify a particular algorithm that defines how job start times are computed from G_0, s, and p. However, it is not hard to see that Definition 5.4.1 only restricts Radermacher's definition to policies that are minimal in the sense that they are not dominated by another preselective policy with the same selection. In fact, Definition 5.4.1 is equivalent to the definition of so-called *MES-policies*, see, e. g., (Igelmund and Radermacher 1983b, Section III); we do not go into details.

If we interpret a preselective policy Π as a function $p \to S(p)$ as defined by (5.2), Π is a composition of minima, maxima, and sums of job processing times. As a direct consequence, Π is monotone and continuous. In fact, Radermacher (1981b) has additionally shown the following, related results.

Theorem 5.4.2. *(Radermacher 1981b, Section II, 3.) 1. Every monotone policy is dominated by a preselective policy. 2. Every continuous and elementary policy is a preselective policy (in the sense of Definition 5.4.1).*

It follows that undesirable effects such as the Graham anomalies do not exist for preselective policies. Moreover, the class of preselective policies is in fact inclusion-maximal with respect to the requirements of a robust policy.

5.4.2 Domination

We next derive a necessary and sufficient dominance criterion for preselective policies. The domination criterion is based on the property that preselected jobs defined for some minimal forbidden sets may additionally solve the conflict on other minimal forbidden sets. We need the following notation. Equivalently to selections we define *partial selections* $s^\ell := (s_1, \ldots, s_\ell)$, $\ell \in \{1, \ldots, f\}$, which

can be seen as selections for the reduced system $\mathcal{F}^\ell := \{F_1, \ldots, F_\ell\}$ of minimal forbidden sets.

Suppose that for a partial selection $s^{\ell-1} = (s_1, \ldots, s_{\ell-1})$ there exists some $j \in F_\ell$ such that each realization $G = (V, E)$ of G_0 and $s^{\ell-1}$ contains a pair $(i, j) \in E$ with $i \in F_\ell \setminus \{j\}$. Then the resource conflict given by F_ℓ never occurs. We say that F_ℓ is *implicitly resolved* by j. The intuition of the dominance criterion is that every extension of the partial selection $s^{\ell-1}$ to a selection s is dominated if not j but some other job $i \in F_\ell \setminus \{j\}$ is chosen as the preselected job for F_ℓ.

Theorem 5.4.3. *A preselected policy Π with selection $s = (s_1, \ldots, s_f)$ is dominated if and only if the resource conflict given by some minimal forbidden set F_ℓ ($\ell = \{1, \ldots, f\}$) is implicitly resolved by some job $j \in F_\ell \setminus \{s_\ell\}$.*

In order to prove Theorem 5.4.3, we again use the concept of waiting conditions. Equivalently to policies we say that a set $\mathcal{W}'$ of waiting conditions dominates another set $\mathcal{W}$ if, for all scenarios $p \in \mathbb{R}^n_>$, the corresponding earliest start schedules $S'(p)$ and $S(p)$ obtained from (5.2) fulfill $S'(p) \leqslant S(p)$. Moreover, recall the following definition from Chapter 2. If for some (U, j) with $j \notin U$ there exists for each realization of $\mathcal{W}$ a pair (i, j) with $i \in U$ we say that (U, j) is *implied* by $\mathcal{W}$. Note that this is particularly the case if $U \supseteq X$ for some $(X, j) \in \mathcal{W}$ (see Section 2.5). For the proof of Theorem 5.4.3 we require the following observations.

Lemma 5.4.4. *Let $\mathcal{W}$ and $\mathcal{W}'$ be feasible sets of waiting conditions. $\mathcal{W}$ is dominated by $\mathcal{W}'$ if and only if each $(X, j) \in \mathcal{W}'$ is implied by $\mathcal{W}$.*

Proof. For an arbitrary scenario $p \in \mathbb{R}^n_>$ let $S(p)$ and $S'(p)$ denote the earliest start schedules obtained by applying (5.2) to $\mathcal{W}$ and $\mathcal{W}'$, respectively. If each $(X, j) \in \mathcal{W}'$ is implied by $\mathcal{W}$ then $S(p)$ must respect at least as many constraints as $S'(p)$ and therefore $S'(p) \leqslant S(p)$.

Let $(X, j) \in \mathcal{W}'$ and suppose that (X, j) is not implied by $\mathcal{W}$. We construct a scenario p as follows. Set $p_i := 1$ if $i \in X$ and $p_i := 0$, otherwise and recall that each (X', j) with $X' \subseteq X$ implies (X, j). We obtain $S'_j(p) = 1$ while $S_j(p) = 0$, and consequently, $\mathcal{W}$ is not dominated by $\mathcal{W}'$. □

Lemma 5.4.5. *If $(U \cup \{h\}, j)$ is implied by a given system $\mathcal{W}$ of waiting conditions with $(U \cup \{j\}, h) \in \mathcal{W}$, then $(U \cup \{h\}, j)$ is also implied by the reduced system $\mathcal{W}' := \mathcal{W} \setminus \{(U \cup \{j\}, h)\}$.*

Proof. Suppose that there exists a realization $G = (V, E)$ of $\mathcal{W}'$ containing no pair $(i, j) \in E$ with $i \in U \cup \{h\}$. We add the pair (j, h) (and transitive pairs) to E. Notice that the resulting relation E' is antisymmetric, since $(h, j) \notin E$ and consequently, (V, E') is a realization of $\mathcal{W}$. However, there exists no pair (i, j)

with $i \in U \cup \{h\}$ which is a contradiction to the fact that $(U \cup \{h\}, j)$ is implied by $\mathcal{W}$. □

Proof of Theorem 5.4.3. Suppose that the resource conflict given by some minimal forbidden set F_ℓ is implicitly resolved by $j \in F_\ell \setminus \{s_\ell\}$. Define the selection s' by $s'_r := s_r$ for $r \in \{1, \dots, f\} \setminus \{\ell\}$ and $s'_\ell := j$. Let $\mathcal{W}$ and $\mathcal{W}'$ denote the sets of waiting conditions induced by s and s'. By assumption, $(F_\ell \setminus \{j\}, j)$ is implied by $\mathcal{W}$. By Lemma 5.4.5 this is also the case for $\mathcal{W} \setminus \{(F_\ell \setminus \{s_\ell\}, s_\ell)\}$. Since $\mathcal{W}' = \mathcal{W} \setminus \{(F_\ell \setminus \{s_\ell\}, s_\ell)\} \cup \{(F_\ell \setminus \{j\}, j)\}$ it follows that each $(X, j) \in \mathcal{W}'$ is implied by $\mathcal{W}$. Therefore, with Lemma 5.4.4, $\mathcal{W}$ is dominated by $\mathcal{W}'$ and it directly follows that Π is dominated by the policy Π' induced by s'.

For the converse, consider a policy Π' with selection s' that dominates Π. For the induced sets of waiting conditions it follows by definition of dominance that $\mathcal{W}'$ dominates $\mathcal{W}$. By Lemma 5.4.4, all $(F_\ell \setminus \{s'_\ell\}, s'_\ell) \in \mathcal{W}'$ ($\ell \in \{1, \dots, f\}$) are implied by $\mathcal{W}$. It thus follows for Π that for all $\ell \in \{1, \dots, f\}$ with $s_\ell \neq s'_\ell$ the resource conflict given by F_ℓ is implicitly resolved by s'_ℓ . □

With the results of Section 2.4, it follows that one can decide in $O(f \cdot (n+m))$ time whether a given preselective policy is dominated or not. This can be done by executing Algorithm 1 for each $F \in \mathcal{F}$ with input $V \setminus F$ and $\mathcal{W}_{V \setminus F}$. The corollary below follows from Corollary 2.4.6.

Corollary 5.4.6. *A minimal forbidden set F is implicitly resolved if and only if, after the execution of Algorithm 1 with input $V \setminus F$ and $\mathcal{W}_{V \setminus F}$, there exists a job $j \in F$ which cannot be added to the list L without violating a waiting condition of $\mathcal{W}$. In this case, j is not executed before at least one $F \setminus \{j\}$ has been completed.*

5.5 Linear Preselective Policies

In this section we introduce the class of *linear preselective policies* which combine the list-oriented features of priority policies with the selection-oriented character of preselective policies. The idea is to define the selection via a priority ordering of the jobs. If L is this ordering, then the preselected job of a minimal forbidden set F is the 'last' job of F in L.

5.5.1 Definition and Characteristics

Definition 5.5.1. *Let Π be a preselective policy with selection $s = (s_1, \dots, s_f)$. Π is called* linear preselective *if there exists a linear extension L of G_0 such that for each minimal forbidden set $F_\ell, \ell \in \{1, \dots, f\}$, the preselected job s_ℓ fulfills $F_\ell \setminus \{s_\ell\} \prec_L s_\ell$, i. e., s_ℓ succeeds all elements of $F_\ell \setminus \{s_\ell\}$ in L.*

Since each linear preselective policy is also a preselective policy, linear preselective policies inherit the analytic properties of being monotone and continuous. We obtain the following characterization of a linear preselective policy.

Theorem 5.5.2. *Let G_0 be a partially ordered set of jobs with an associated system $\mathcal{F}$ of minimal forbidden sets $\{F_1, \ldots, F_f\}$.*

(1) Every linear extension of G_0 induces a linear preselective policy.

(2) A preselective policy Π with selection $s = (s_1, \ldots, s_f)$ is linear preselective if and only if the digraph D defined by the set $\mathcal{W}$ of waiting conditions that are induced by s and E_0 is acyclic.

Proof. We only give a proof of claim (2), (1) follows immediately from the definition of linear preselective policies. Let an ordering L represent a linear preselective policy. We extend the ordering to an ordering of all nodes of D by inserting the OR-node which represents a minimal forbidden set F directly before the last job of F. Then L defines a topological ordering of the nodes of D and thus D is acyclic. Conversely, let D be acyclic and let L be an ordering of the jobs obtained from a topological sort of the nodes of D by ignoring the OR-nodes that represent minimal forbidden sets. It follows from the definition of D that $X \setminus \{j\} \prec_L j$ for all waiting conditions (X, j) represented by D. Consequently, the preselected job j of each minimal forbidden set F fulfills $F \setminus \{j\} \prec_L j$. Furthermore, since each precedence constraint (i, j) is modeled by a waiting condition $(\{i\}, j)$, L is an extension of G_0. Therefore, the preselective policy corresponding to D is linear preselective. □

Let us consider Example 1.3.1 and the linear preselective policy defined by the sequence $L = 3 \prec 1 \prec 5 \prec 2 \prec 4$. The resulting selection is $s = (5, 4, 4)$ and for the scenario $p = E[\boldsymbol{p}]$ we obtain $S(p) = (0, 0, 0, 5, 3)$.

Next, we discuss several aspects that are direct consequences of the definition of linear preselective policies.

Computing Earliest Job Starting Times. In Section 5.4.1 we noted that, for a preselective policy and a given scenario p, earliest job start times can be computed in $\mathrm{O}(n + m + f \log f)$ by Algorithm 3. For linear preselective policies, we can compute earliest job start times more efficiently in $\mathrm{O}(\, n+m\,)$ time. We exploit the fact that the selection s of such a policy can be represented by a linear extension L of the underlying partial order of precedence constraints by calculating the start times in the order of L. Thus, no heap is required which yields the improved running time. A pseudo-code of this simple procedure is given in Algorithm 7.

Algorithm 7: Computing earliest start times for a linear preselective policy

Input : A directed acyclic graph D representing a set V of jobs and waiting conditions $\mathcal{W}$ with positive arc weights on the arcs in $V \times \mathcal{W}$.

Output: A minimal schedule $S \in \mathbb{R}_{\geqslant}^{|\mathcal{V}|}$.

for $j \in \mathcal{V}$ *along a topological ordering in* D **do**
  if j *is an* AND*-node* **then**
    $S_j := \max_{i \in in(j)} S_i$;
  else $S_j := \min_{i \in in(j)}(S_i + p_i)$;
return S;

Memory Requirements. Another computational benefit of linear preselective policies is their compact representation. In contrast to preselective policies, every linear preselective policy can be stored within $\mathrm{O}(n)$ space by the linear ordering L of the jobs which defines the policy. This is particularly helpful in the context of branch-and-bound approaches where typically a large number of policies has to be maintained for the same set $\mathcal{F}$ of minimal forbidden sets.

5.5.2 Domination

We now show that 'quite a few' preselective policies that are dominated are not linear preselective. Thus, if we deal with linear preselective policies, all these dominated policies are discarded from consideration beforehand. To make this intuition more precise, consider a partial selection $s^{\ell-1} = (s_1, \ldots, s_{\ell-1})$ inducing a set $\mathcal{W}$ of waiting conditions which imply the waiting condition $(F_\ell \setminus \{j\}, j)$. Defining j as the preselected job for F_ℓ thus creates no further restrictions. Contrarily, if we choose a job $i \in F_\ell$ to be preselected for F_ℓ but $(F_\ell \setminus \{i\}, i)$ is not implied by $\mathcal{W}$, we obtain an additional waiting condition $(F_\ell \setminus \{i\}, i)$. Thus, any such preselective policy is dominated.

Corollary 5.5.3. *Let $s^{\ell-1}$, $\mathcal{W}$, and $j \in F_\ell$ be as above. Denote by $U \subseteq F_\ell \setminus \{j\}$ the set of jobs i such that there exists a directed path from i to some other job $h \in F_\ell \setminus \{i\}$ in the digraph D resulting from $\mathcal{W}$. Then all preselective policies that are induced by extending $s^{\ell-1}$ to a selection s with $s_\ell \in U$ are dominated and not linear preselective.*

Proof. It follows from Theorem 5.4.3 that Π is dominated if $s_\ell \neq j$. This particularly holds for all $s_\ell \in U$. However, choosing s_ℓ from U would induce a cycle in D and thus, by Theorem 5.5.2, Π is not linear preselective. □

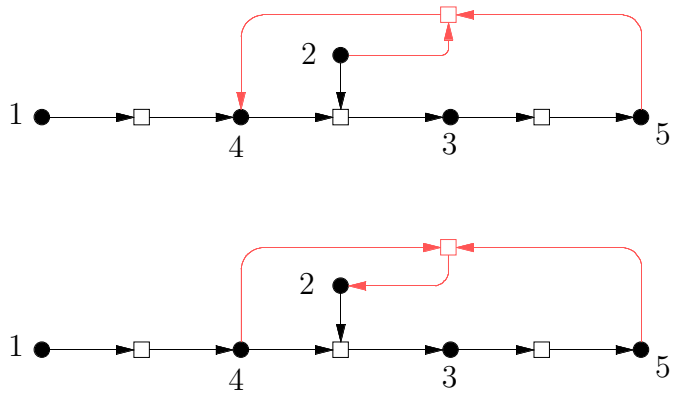

Figure 5.5: The figure shows the digraphs of waiting conditions which results from Example 1.3.1 and the preselection of job 3 with respect to the minimal forbidden set $\{2,3,4\}$ (black components only). The additional (grey) waiting conditions represent the minimal forbidden set $\{2,4,5\}$ with preselected jobs 4 and 2, respectively. Both choices yield a dominated policy because $\{2,4,5\}$ is implicitly resolved by job 5. In the linear preselective case, in all orderings L of jobs which allow job 3 to be preselected with respect to $\{2,3,4\}$ we have $2 \prec_L 5$ and $4 \prec_L 5$ and thus, preselecting job 5 for $\{2,4,5\}$ is the only possible choice.

In Example 1.3.1, there exist $2 \cdot 3 \cdot 3 = 18$ selections, including one infeasible selection, see (Igelmund and Radermacher 1983b, Figure 5) for details. As it can easily be verified, 11 of the 17 policies that are induced from the feasible selections are dominated. Among these dominated policies, only one is a linear preselective policy while all non-dominated policies are linear preselective. The example is further discussed in Figure 5.5.

5.5.3 Acyclic Preselective Policies

We next define the class of *acyclic preselective policies* which extends the class of linear preselective policies. Let $s^{\ell-1}$ be a partial selection which yields an acyclic digraph of waiting conditions. As in the previous section, let the minimal forbidden set F_ℓ be implicitly resolved by the waiting condition $(F_\ell \setminus \{j\}, j)$ with respect to $s^{\ell-1}$. Then, a preselective policy which is induced by extending $s^{\ell-1}$ to a selection s with $s_\ell = j$ is not necessarily linear preselective. We demonstrate this observation by the following example.

Example 5.5.4. *Let $G_0 = (V, E_0)$ be given by $V = \{1,2,3,4,5\}$ and $E_0 = \emptyset$ and let the sets $F_1 := \{1,2\}$, $F_2 := \{2,3\}$, $F_3 := \{3,4,5\}$, and $F_4 := \{1,3,5\}$ be minimal forbidden.*

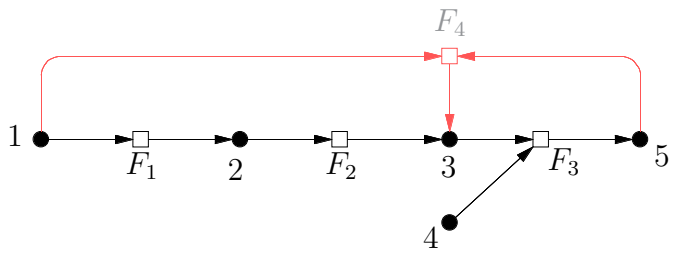

Figure 5.6: The figure shows the digraphs of waiting conditions which results from Example 5.5.4 and the selections $s = (2, 3, 5)$ (black components only) and $s' = (2, 3, 5, 3)$ (black and grey components), respectively. While the digraph representing s is acyclic, we obtain a cyclic digraph for s'.

Consider the (linear preselective) partial selection $s = (2, 3, 5)$; the resulting digraph of waiting conditions is given in Figure 5.6. Then the minimal forbidden set F_4 is implicitly resolved because job 3 never starts before job 1 has been completed. If job 3 is explicitly chosen as the preselected job for F_4, we obtain a cycle in the associated digraph of waiting conditions and consequently, the resulting selection is not linear preselective. The only extension of s which leads to a linear preselective policy is to choose job 5 as the preselected job for F_4. But this results in the superfluous waiting condition $(\{1, 3\}, 5)$.

The above observation suggests to consider an alternative class of scheduling policies which is closely related to the class of linear preselective policies.

Definition 5.5.5. *A preselective policy* Π *is an* acyclic preselective policy *if the set* $\mathcal{W}$ *of waiting conditions that defines* Π *yields an acyclic digraph.*

On a first glance, the classes of acyclic preselective policies and linear preselective policies appear to coincide, however, this in not the case. Clearly, each linear preselective policy is also an acyclic preselective policy. To see that the reverse is not valid, observe that, by definition, a linear preselective policy has a preselected job for each minimal forbidden set F, even if F is implicitly resolved. This is not the case for an acyclic preselective policy. Moreover, in a linear preselective policy, the preselected job uniquely determines the waiting condition $(F \setminus \{j\}, j)$ which guarantees that the jobs in F are not processed simultaneously. For an acyclic preselective policy, if it explicitly resolves F, every subset X of $F \setminus \{j\}$ may be the predecessor set of the waiting condition. Clearly, a larger predecessor set yields more flexibility in general, hence $F \setminus \{j\}$ is the most appropriate predecessor set. However, as Example 5.5.4 demonstrates, if we restrict to sets of waiting conditions that yield *acyclic* digraphs this needs not necessarily be the case.

We further discuss the relationship of linear preselective policies and acyclic preselective policies in Sections 5.7 and 6.3.3 below.

5.6 Job-Based Priority Policies

We next study a subclass of the class of preselective policies that has an important benefit compared to the classes of policies discussed in Sections 5.3–5.5: They do not require the representation of resource constraints by (possibly exponentially many) minimal forbidden sets. Like linear preselective policies, *job-based priority policies* can be represented by an ordering of the jobs. However, the algorithm to transform a given scenario $p \in \mathbb{R}^n_>$ into a schedule is different. In the order of L, they start every job as early as possible with the side constraint that $S_i(p) \leqslant S_j(p)$ if $i \prec_L j$. This 'job-based' view instead of the (greedy) 'resource-based' view for priority policies based on Graham's list scheduling is the reason for their name. Job-based priority policies have previously been used within different stochastic and deterministic scheduling models. For instance, they have been applied to derive approximation algorithms for stochastic machine scheduling problems (Möhring, Schulz, and Uetz 1999) and also within branch-and-bound procedures for deterministic resource-constrained project scheduling (Sprecher 2000).

5.6.1 Definition and Characteristics

Before we give a definition of a job-based priority policy, let us formulate the algorithm that maps a given scenario p to a schedule. For a given ordering L and given scenario p, we start each job in the order of L as early as possible (with respect to precedence and resource constraints) but not before the start time of some previously started job. Consequently, for some $j \in V$, we traverse the schedule constructed so far over time and set S_j to the smallest time that is both precedence- and resource-feasible. The traversal is started at the start time of the direct predecessor of j in L. An implementation of this coarse description is given in Algorithm 8 which runs in $O(\, n|K| + |E_0| + n \log n \,)$ time. The definition of a job-based priority policy now reads as follows.

Definition 5.6.1. *A policy* Π *(as in Definition 5.2.1) is called* job-based priority policy *if there is an ordering* L *of the jobs which extends* E_0 *such that, for all* $j \in V$ *and* $p \in \mathbb{R}^n_>$, $S^{\Pi}_j(p)$ *equals the start times obtained from Algorithm 8.*

Notice that neither the combinatorial representation of a job-based priority as a job ordering nor Algorithm 8 requires the forbidden set representation of resource constraints. This particularly allows to apply such policies to (large) projects

Algorithm 8: Computing earliest start times for a job-based priority policy

Input : Set V of jobs, precedence constraints E_0, resource-constraints in threshold representation, job ordering L which extends E_0.

Output: A feasible schedule S.

```
ES := 0;                                   // ES is a vector of size |V|
t := 0;
CurrR := 0;                    // CurrR, R, and r_j are vectors of size |K|
for j ∈ V in the order of L do
    CurrR := CurrR + r_j;
    while (t < ES_j) or (CurrR_k > R_k for some k) do
        t := smallest completion time C_h of some job h with C_h > t;
        for i ∈ V with C_i = t do
            CurrR := CurrR − r_i;
    S_j := t;
    for all successors i of j in G_0 do
        ES_i := t + p_j;
return S;
```

where the (possibly exponential) number of minimal forbidden sets makes the use of forbidden set based policies computationally inefficient. When the job-based priority policy $L = 3 \prec 1 \prec 5 \prec 2 \prec 4$ is applied to Example 1.3.1 with $p = E[\boldsymbol{p}]$ we obtain $S(p) = (0, 3, 0, 8, 3)$. Note that this schedule differs from the schedule that is obtained if the ordering L is interpreted as a linear preselective policy where $S(p) = (0, 0, 0, 5, 3)$.

5.6.2 Domination

We next study the relationship between a job-based priority policy and a linear preselective policy that are induced by the same ordering.

Theorem 5.6.2. *If a job-based priority policy* Π *and a linear preselective policy* Π' *are induced by the same linear ordering* L *of jobs, then* Π' *dominates* Π.

Proof. Consider a minimal forbidden set F and let j be its preselected job with respect to Π', i. e., $F \setminus \{j\} \prec_L j$. Let $p \in \mathbb{R}^n_>$ be an arbitrary scenario. Since $S^{\Pi}_i(p) \leqslant S^{\Pi}_j(p)$ for all $i \prec_L j$ this also holds for all $i \in F \setminus \{j\}$. Consequently, since S^{Π} is feasible, we have $S^{\Pi}_j(p) \geqslant \min_{i \in F \setminus \{j\}}(S^{\Pi}_i(p) + p_i)$. It follows that Π induces at least as many constraints as Π' and thus, Π' dominates Π. □

We next argue that job-based priority policies fulfill the requirements of a robust policy of being monotone and continuous. We show that any job-based priority policy can be expressed as a set of waiting conditions. Let Π and Π' be as in Theorem 5.6.2 and consider the set $\mathcal{W}'$ of waiting conditions resulting from Π'. We iteratively construct a set $\mathcal{W} \supseteq \mathcal{W}'$ such that the preselective policy that is induced from $\mathcal{W}$ and the job-based priority policy Π coincide. First, set $\mathcal{W} := \mathcal{W}'$. Then, for each j in the order of L add all (X, j) to $\mathcal{W}$ with $(X, i) \in \mathcal{W}$ (where i denotes the job which immediately precedes j in L). It is not hard to see that, for each p, the earliest start schedule obtained from $\mathcal{W}$ according to (5.2) and the schedule S resulting from Π coincide. Hence, we obtain the following result.

Theorem 5.6.3. *Let G_0 be a partially ordered set of jobs with an associated system $\mathcal{F}$ of forbidden sets. If Π is a job-based priority policy for G_0 and $\mathcal{F}$ then Π is linear preselective for G_0 and a larger system $\mathcal{F}' \supseteq \mathcal{F}$ of forbidden sets.*

As a direct consequence of Theorem 5.6.3, if we interpret a job-based priority policy Π as a function $p \to S^{\Pi}(p)$, Π is monotone and continuous and consequently, Graham-anomalies do not occur within the class of job-based priority policies.

5.7 Relationship between Optimum Values

We now relate the different classes of policies with respect to their optimum expected project cost ρ. To simplify notation we introduce the following identifiers for the classes of policies that have been mentioned in this chapter. We denote the classes of preselective, acyclic preselective, and linear preselective policies by PRS, ACY, and LIN, respectively. The class of ES-policies is abbreviated by ES. The (resource-based) priority policies based on Graham's list scheduling are referred to as RBP and the class of job-based priority policies is denoted by JBP.

If processing times are deterministic then, except for priority policies (RBP), each of the above classes of policies achieves the deterministic optimum value, i. e., $\rho^{\text{PRS}} = \rho^{\text{ACY}} = \rho^{\text{LIN}} = \rho^{\text{ES}} = \rho^{\text{JBP}}$. To see this, construct an appropriate policy Π from an optimal schedule S^* and show that Π yields a schedule S with no larger cost.

In the stochastic case, however, the behavior is quite different. According to the definitions and results presented in the previous sections we immediately obtain $\rho^{\text{PRS}} \leqslant \rho^{\text{ACY}} \leqslant \rho^{\text{LIN}} \leqslant \rho^{\text{JBP}}$. Moreover, the class of acyclic preselective policies always contains a policy that performs as good as an optimal ES-policy. To see this, observe, that an extension $G = (V, E)$ which represents an optimal ES-policy can directly be transformed into a set of waiting conditions which can be represented by an acyclic digraph. Hence we obtain $\rho^{\text{ACY}} \leqslant \rho^{\text{ES}}$. Note that this

construction is not valid for linear preselective policies. In fact, the classes of ES-policies and linear preselective policies are incomparable. It is easy to construct an instance with $\rho^{\mathrm{LIN}} < \rho^{\mathrm{ES}}$; for instance, Example 1.3.1 is appropriate. An instance with $\rho^{\mathrm{LIN}} > \rho^{\mathrm{ES}}$ is given next. It is based on the observation that linear preselective policies explicitly define a preselected job for all minimal forbidden sets although some of them may be implicitly resolved. This may result into superfluous waiting conditions (recall Section 5.5.2).

Example 5.7.1. *Let $V = \{1, \ldots, 8\}$ and $E_0 = \{(2,6),(3,7),(4,8)\}$. Minimal forbidden sets are given by $F_1 = \{1,6\}$, $F_2 = \{6,7\}$, $F_3 = \{5,7,8\}$, and $F_4 = \{1,7,8\}$. The job processing times are as follows: $p_1 \in \{1,15\}$, $p_2 = 8$, $p_3 = 12$, $p_4 = 14$, $p_5 = 14$, $p_6 = 4$, $p_7 = 2$, $p_8 \in \{1,15\}$. Only the processing times of jobs 1 and 8 are random, each value appears with probability $\frac{1}{2}$. The distributions are assumed to be independent.*

The unique optimal ES-policy has expected makespan 23.5 while the value of the best linear preselective policy $s = (6,7,8,8)$ is 23.75. We obtain $\rho^{\mathrm{ES}} = 23.5 < 23.75 = \rho^{\mathrm{LIN}}$. The difference is due to the waiting condition $(\{1,7\},8)$. This waiting condition is superfluous since F_4 is implicitly resolved due to the choices of preselected jobs for F_1 and F_2. Note that Example 5.7.1 particularly implies that $\rho^{\mathrm{ACY}} < \rho^{\mathrm{LIN}}$ is possible.

Let us next compare the classes of preselective policies and acyclic preselective policies. In fact, there are instances with $\rho^{\mathrm{PRS}} < \rho^{\mathrm{ACY}}$, however, each of the two instances we present next is rather artificial. The first instance has dependent processing time distributions. For the second instance we use the objective to minimize the total weighted completion time of jobs. There, job weights w_j must be chosen carefully in order to achieve the desired property ($\rho^{\mathrm{PRS}} < \rho^{\mathrm{ACY}}$). It is an open question whether there are instances with this property and less degrees of freedom, such as independent processing time distributions in conjunction with the cost functions C_{max} or $\sum_j C_j$.

Example 5.7.2. *Let $V = \{1,2,3,4\}$, $E_0 = \emptyset$, $F_1 = \{1,2,3\}$, and $F_2 = \{1,2,4\}$. Furthermore, the following four scenarios of processing times are possible, each with probability $\frac{1}{4}$. $p^1 = (2,2,1,5)$, $p^2 = (2,2,5,1)$, $p^3 = (6,2,1,5)$ and $p^4 = (2,6,5,1)$.*

A straightforward case analysis shows that the non-linear preselective selection $s = (1,2)$ is the unique optimal solution in the class of preselective policies. We obtain $\rho^{\mathrm{PRS}} = 6 < 6.25 = \rho^{\mathrm{ACY}}$. Note that this instance involves stochastically dependent job processing times. We next provide another counterexample where job processing times are independent and where the objective is to minimize the weighted sum of completion times.

Example 5.7.3. *Let* $V = \{1, 2, 3, 4\}, E_0 = \emptyset$*, and* $F_1 = \{1, 2, 3\}$*,* $F_2 = \{1, 2, 4\}$*,* $F_3 = \{1, 3, 4\}$*, and* $F_4 = \{2, 3, 4\}$*. Each job processing time is exponentially distributed with rates* $\lambda_1 = 10^6$*,* $\lambda_2 = 1$*,* $\lambda_3 = 10$*, and* $\lambda_4 = 1$*, respectively. The job weights are* $w_1 = 10^6$*,* $w_2 = 10^6$*,* $w_3 = 2$*,* $w_4 = 10$*.*

Kämpke (1985) has shown that the unique optimal scheduling policy for this 2-machine instance is to first schedule jobs 1 and 2 and then, if job 1 completes before job 2, plan according to priority ordering $3 \prec 4$, otherwise use the priority ordering $4 \prec 3$. Observe that this scheduling policy is equivalent to a preselective policy Π with selection $s = (3, 4, 3, 4)$. Since Π is not acyclic preselective, it follows that $\rho^{\text{PRS}} < \rho^{\text{ACY}}$.

We next discuss the relation between the classes of ES-policies and job-based priority policies. In fact, it can verified by simple counter-examples that the classes ES and JBP are not comparable in terms of their optimum value. An analysis of Example 1.3.1 yields $\rho^{\text{ES}} < \rho^{\text{JBP}}$. An instance with $\rho^{\text{ES}} > \rho^{\text{JBP}}$ is as follows.

Example 5.7.4. *Let* $V = \{1, 2, 3\}, E_0 = \emptyset$*, and* $F = V$*. Each job processing time is* $p_i \in \{1, 2\}$*,* $i \in \{1, 2, 3\}$*; each value appears with probability* $\frac{1}{2}$*. The distributions are assumed to be independent.*

Then, the pair $(2, 3)$ defines an optimal ES-policy with expected makespan 3. On the other hand, the job-based priority policy that is defined by the ordering $L = 1 \prec 2 \prec 3$ yields a value of 2.75.

To summarize the analysis of Example 1.3.1 we obtain the ordering $\rho^{\text{PRS}} = \rho^{\text{ACY}} = \rho^{\text{LIN}} < \rho^{\text{ES}} < \rho^{\text{JBP}}$. In fact, as will be shown in the computational study in Chapter 6 below, we empirically obtain the same ordering for the entire test set we used within our computational experiments.

It remains to discuss the relation of optimum values of the above discussed, monotone and continuous classes of policies with the class of traditional priority policies. Here, it is easy to see that each of the classes is incomparable to the class RBP. For Example 5.2.3 with deterministic $p' = (3, 1, 1, 4, 4, 9, 9)$, there exists no priority policy which yields the optimal deterministic makespan. Hence, we obtain $\rho^{\text{JBP}} < \rho^{\text{RBP}}$. On the other hand, it is trivial to construct an instance with $\rho^{\text{RBP}} < \rho^{\text{PRS}}$. For Example 5.7.5 below we obtain $\rho^{\text{PRS}} = \rho^{\text{LIN}} = \rho^{\text{ES}} = \rho^{\text{JBP}} = 3.5$ while $\rho^{\text{RBP}} = 3.25$.

Example 5.7.5. *Let* $V = \{1, 2, 3, 4\}, E_0 = \{(1, 3)(2, 4)\}$*, and* $F = \{3, 4\}$*. Job processing times are* $p_1 \in \{1, 2\}$*,* $p_2 \in \{1, 2\}$*,* $p_3 = p_4 = 1$*; Only the processing times of jobs* 1 *and* 2 *are random, each value appears with probability* $\frac{1}{2}$*. The distributions are assumed to be independent.*

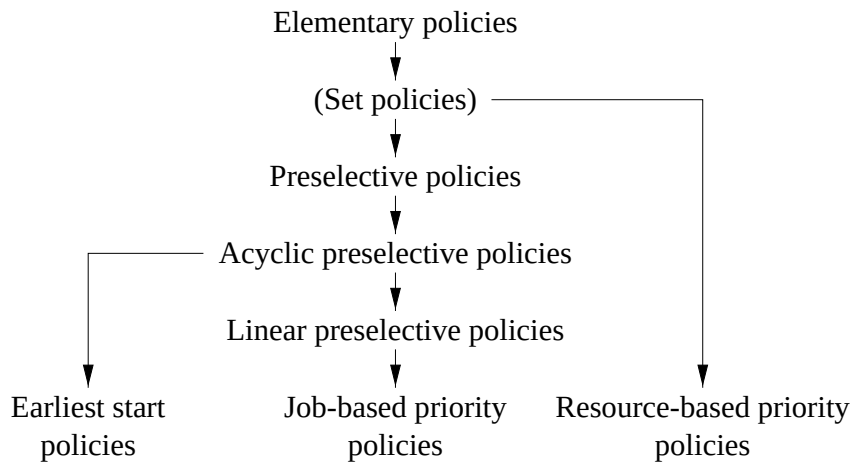

Figure 5.7: The graph indicates the relationship of different classes of scheduling policies with respect to their optimum value ρ. A directed path from class A to B indicates $\rho^A \leqslant \rho^B$. To be complete, we have included the class of *set policies* which has not been mentioned in the chapter. For a definition of set policies as well as various related results we refer to (Möhring, Radermacher, and Weiss 1985).

In Figure 5.7, we summarize the discussion on the optimum values by a graphic representation of the different classes of policies. Figure 5.7 complements a similar figure given in (Möhring and Radermacher 1985; Radermacher 1986).

CHAPTER 6

BRANCH-AND-BOUND ALGORITHMS FOR STOCHASTIC RESOURCE-CONSTRAINED PROJECT SCHEDULING

In this chapter we address computational issues for solving stochastic resource-constrained project scheduling problems. We apply the results of the previous chapter and develop different branch-and-bound algorithms for the problem. The objective function we want to minimize is the project makespan in expectation. The purpose of the study is twofold. First, we establish results on the trade-off between computational efficiency and solution quality for the classes of preselective and linear preselective policies as well as for job-based priority policies and ES-policies. Second, we develop and apply various ingredients such as dominance rules and lower bounds that turn out to be useful within the computation. In order to comprehensively study these issues we have implemented five different branch-and-bound algorithms and explore their computational behavior on 1440 test instances.

6.1 Introduction and Related Work

We already noted in Chapter 1 that the literature is extensive in regard to computational issues related to both PERT problems and deterministic resource-constrained project scheduling problems. However, their combination, the stochastic resource-constrained project scheduling problem has been largely ignored. Most relevant for our study is the work of Igelmund and Radermacher (1983a) who develop a branch-and-bound algorithm in order to compute optimal preselective policies. To the best of our knowledge, this is the only reference in the direction of computing a policy which is optimal within a particular class. Their branch-and-bound algorithm is based on systematically resolving minimal forbidden sets in order to enumerate all selections of a given instance. We give more details on the branching process in Section 6.2 below. Their computational experiments are limited to few small instances which comprise 16 jobs and 7 minimal forbidden sets.

All other computational contributions have been established very recently;

none of them is based on the concepts that were developed by Igelmund and Radermacher (1983a, 1983b). Instead, they all consider (resource-based) priority policies. Let us briefly review the recent work.

The NASA (Henrion, Fung, Cheung, Steele, and Basevich 1996) has set up a research project on stochastic resource-constrained project scheduling. The main intention was to develop a software for supporting an integrated schedule and cost risk analysis for space shuttle ground processing (the resulting tool became later known as 'SCRAM'). Most relevant from the theoretical perspective is a heuristic method to compute *critical* jobs. If no resource constraints have to be respected and job processing times are deterministic, jobs are called *critical* if they have a *slack* of 0. For given earliest and latest possible start time ES_j and LS_j of a job $j \in V$, respectively, the *slack* of j is defined as $LS_j - ES_j$ (here, each LS_j results from the fictive deadline ES_b). Recall that b is a dummy job which marks the project completion. Following an early work of Wiest (1963), Henrion et al. (1996) formulate a generalization of the notion of *slack* which covers the presence of limited resources. For a given scenario p their approach basically is to first compute job start times $S_j(p)$ with respect to some priority-driven heuristic A. (They refer to A as 'the typical heuristic resource leveling algorithm'). Then they define new job priorities according to decreasing $C_j(p)$ and, with these priorities, apply A to the instance that is obtained from reversing all pairs of E_0. This yields another schedule S'. Then, $S_j(p)$ and $S_b(p) - S'_j(p) - p_j$ are interpreted as the earliest start and the latest start of job j, respectively and $S_b(p) - S'_j(p) - p_j - S_j(p)$ is defined as the slack of j. Henrion et al. (1996) use simulation in order to generate a set of 400 scenarios p from given job distributions (we noted the basic methodology of simulation in Section 1.2). Then, the above method is applied to each scenario. Based on the resulting data, a distribution for the slack is created. We like to point out that, if the used algorithm A is a resource-based priority policy, i. e., Graham's list scheduling, then the greedy behavior of A may force some jobs to start early (late) with respect to *both* schedules $S(p)$ and $S'(p)$. This may yield even negative slack values (for appropriately constructed instances) which is certainly not a desired output.

Golenko-Ginzburg and Gonik (1997) suggest a dynamic (resource-based) priority policy in order to compute a feasible solution to the stochastic resource-constrained project scheduling problem. Here, *dynamic* means that the job priorities are not constant; for a given decision time they may depend on the past. The cost function which Golenko-Ginzburg and Gonik (1997) consider is to minimize the expected makespan. At each job completion time t (initially $t = 0$), their algorithm first computes for each job j that is not yet scheduled the probability q_j that j is on a critical path when all resource conflicts after time t are neglected. The q_j are approximated by simulation. Next, among the set of jobs B that are precedence-feasible at t, a subset $B' \subseteq B$ of jobs is started at t with the prop-

erty that $\sum_{j \in B'} r_{jk} \leqslant R_k$ for all k and $\sum_{j \in B'} q_j$ is maximized. Notice that, for $|K| = 1$, this is the classical Knapsack problem, hence, to obtain B', Golenko and Gonik solved an NP-hard optimization problem at each job completion. They also suggest to heuristically compute the set B'. The approach is tested on a single instance (though different processing time distributions are considered). We give results of our algorithms on that instance in Section 6.5.5 below.

Tsai and Gemmill (1998) have implemented a Tabu-Search heuristic in order to compute (resource-based) priority policies for the problem of minimizing the expected makespan. The neighborhood they use is based on interchanging two randomly chosen jobs in the list L which represents the priority policy. Then, each generated policy is evaluated by 100 scenarios that were drawn from the job processing time distributions by simulation. To evaluate the behavior of their algorithm, Tsai and Gemmill consider the classical 110 instances collected by Patterson (1984) and generated a beta distribution $\boldsymbol{p}_j$ for each job. On average, their results are roughly 2.5% larger than the deterministic makespan (with respect to $E[\boldsymbol{p}]$). They allowed a computation time of roughly 10 seconds per instance on a Pentium-166 computer. The paper also contains results for three instances that were adopted from an aircraft maintenance facility, however, according to Gemmil (2000), they are not allowed to circulate these instances.

Valls, Laguna, Lino, Pérez, and Quintanilla (1998) consider resource-constrained project scheduling problems with stochastic job interruptions. That is, it may happen that jobs are interrupted for an uncertain amount of time. In their model, each job consists of three parts: The processing time of the first part is deterministic, the second part models the job interruption for a time period of uncertain length, and the third part represents the job after the interruption (the processing time of the third part is assumed to be random as well). The model allows only one interruption per job. While jobs are interrupted, the allocated resources are freed and may be used to process other jobs. However, notice that for some job j, this characteristic can be modeled by three jobs $h \prec_{E_0} i \prec_{E_0} \ell$ that represent the three parts of j. The processing time p_h of h is deterministic and $\boldsymbol{p}_i$ and $\boldsymbol{p}_\ell$ are random according to the distributions of the second and the third part of j. Finally, set $r_h := r_\ell := r_j$ and $r_i := 0$. Valls et al. (1998) consider the objective of minimizing the total weighted tardiness in expectation. Based on (resource-based) priority policies, they develop a local search heuristic which basically follows the framework of a genetic algorithm. Before the local search is applied, as a starting point, an initial set of policies is computed by different construction schemes. Like Tsai and Gemmill (1998) they modify job orderings (which represent the priority policies) in order to find new solutions. The cost of each policy is approximately computed by simulation. For their computational testings they generate 324 instances with $50 - 150$ jobs and use 30 scenarios of job processing times. The cost of the best policy they found by the local search

is considerably lower when compared to the best solution within the initial set of solutions. The required computation times vary from roughly ten minutes up to two hours on a Pentium-166 computer.

The major drawback of each of the discussed attempts to solve stochastic resource-constrained project scheduling problems is that they are based on a class of priority policies that show Graham-anomalies. We therefore do not follow their line of research. Instead, based on the work of Igelmund and Radermacher (1983a), we develop a branch-and-bound procedure to compute an optimal preselective policy (based on the set of scenarios we consider). In contrast to Igelmund and Radermacher (1983a), however, we employ the results of the previous chapters to improve the performance of our algorithm. In particular, we make use of the efficient algorithm to estimate the expected makespan of preselective policies (see Section 5.4). Moreover, we apply dominance rules that may prune large portions of the search tree (see Section 5.4.2 and Section 6.3 below). In addition, featuring two different branching schemes, we have implemented two branch-and-bound algorithms for linear preselective policies and one algorithm for each of the classes of ES-policies and job-based priority policies. We explore their computational efficiency on 1440 instances of different size that have been generated with the instance generator ProGen (Kolisch and Sprecher 1996).

The chapter is organized as follows. In Section 6.2 we motivate the usefulness of studying branch-and-bound algorithms for stochastic resource-constrained project scheduling and describe the used branching schemes. We then state a dominance rule for each of the branch-and-bound algorithms in Section 6.3. Section 6.4 gives a more detailed account of some ingredients that helped to speed up the computations. Our computational study is presented in Section 6.5.

6.2 Branch-and-Bound and Random Processing Times

In Chapter 1.1 we mentioned several alternatives how the search for an optimal solution by a branch-and-bound can be organized for a deterministic resource-constrained project scheduling problem. In this section we describe two such approaches that we use to compute an optimal policy within the stochastic setting. In addition to that we note how we compute the expected makespan for partial solutions that are constructed within the branch-and-bound algorithms. Before, however, let us briefly emphasize that preselective policies Π which are optimal with respect to the expected job processing times $E[\boldsymbol{p}]$, do not yield an optimum policy for the stochastic problem with processing times $\boldsymbol{p}$ in general. This is shown by the following example.

Example 6.2.1. *Let $G_0 = (V, E_0)$ be given by $V = \{1, 2, 3, 4, 5, 6\}$ and $E_0 = \{(1, 4), (1, 5), (2, 5), (3, 5), (3, 6)\}$ and let the set $F = \{1, 2, 3\}$ be minimal for-*

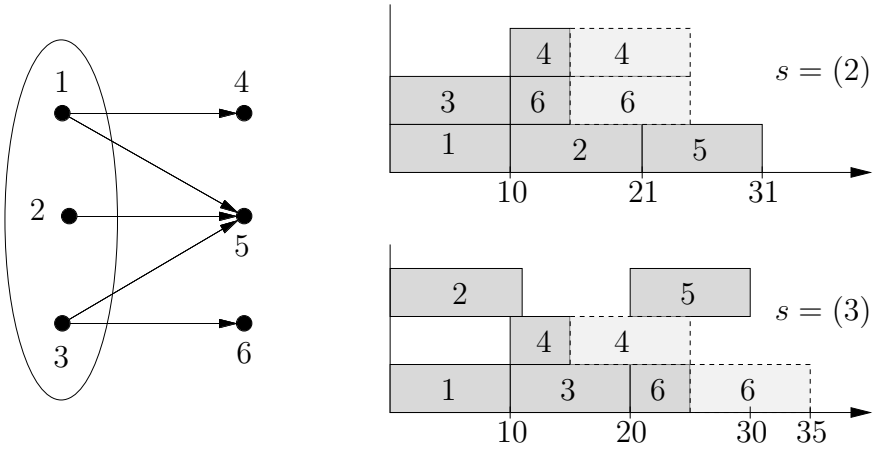

Figure 6.1: The instance given in Example 6.2.1 together with Gantt charts for the selections $s = (2)$ and $s = (3)$ (the selection $s = (1)$ is symmetric to $s = (3)$). Both Gantt charts show the different processing times of the jobs 4 and 6. While $E[C_{max}(\boldsymbol{p})] = 31$ for $s = (2)$ we obtain $E[C_{max}(\boldsymbol{p})] = 32.5$ for $s = (1)$ and $s = (3)$. For deterministic processing times $E[\boldsymbol{p}]$ we obtain $C_{max}(E[\boldsymbol{p}]) = 31$ for $s = (2)$ and $C_{max}(E[\boldsymbol{p}]) = 30$ for $s = (1)$ and $s = (3)$.

bidden. The following processing times are deterministic: $p_1 = 10, p_2 = 11, p_3 = 10, p_5 = 10$. *The processing times of jobs* 4 *and* 6 *are independently distributed with* $Pr(p_4 = 5) = Pr(p_4 = 15) = \frac{1}{2}$ *and* $Pr(p_6 = 5) = Pr(p_6 = 15) = \frac{1}{2}$.

The distributions of job processing times yield 4 different scenarios. It can easily be evaluated that the selections $s = (1)$ and $s = (3)$ lead to an optimal policy for the deterministic problem with processing times $E[\boldsymbol{p}]$ and $s = (2)$ is the unique optimal policy in the stochastic case (see Figure 6.1). Thus, if the objective is to find an *optimal* preselective policy, it is not sufficient to compute all policies which are optimal with respect to the expected job processing times $E[\boldsymbol{p}]$ and then, among these policies, choose one with smallest expected makespan.

Branching schemes. For a given instance I a branch-and-bound algorithm (implicitly) enumerates a set $\mathcal{S}$ of feasible solutions which contains at least one optimal solution of I. The set $\mathcal{S}$ is usually organized in a rooted tree (the so-called *search-tree*) where the root node represents the given instance I and the set of leaves represents $\mathcal{S}$. Each internal node of the tree represents an appropriately constructed sub-instance of I. The most essential characteristic of a branch-and-bound algorithm is certainly the *branching scheme* that defines how a given (sub)instance is split into a set of sub-instances.

We employ two different branching schemes that have previously been proposed in the literature, we call them the *forbidden set branching scheme* and the

precedence-tree branching scheme. In the latter scheme all linear extensions of G_0 are enumerated. In the *forbidden set branching scheme*, for each $F \in \mathcal{F}$, all alternatives to resolve F are enumerated. We apply the forbidden set branching scheme to preselective policies, linear preselective policies, and ES-policies and the precedence-tree branching scheme to linear preselective policies and job-based priority policies. Since, for linear preselective policies, it is not a priori clear which branching scheme is superior we have implemented both alternatives which in total leads to five different branch-and-bound algorithms.

We first describe the *forbidden set branching scheme*. We assume that the minimal forbidden sets are given in a fixed order (we discuss appropriate orders in Section 6.4.4 below). Each node v in the search tree is associated with a minimal forbidden set F and branching on v systematically resolves F. For ES-policies we create a child node u_{ij} of v for each ordered pair $(i,j), i,j \in F, i \neq j$. For preselective and linear preselective policies we create a child node u_j of v for each $j \in F$. Then, each leaf v of the search tree represents a policy which is defined by resolving each minimal forbidden set according to the decisions made on the path from v to the root of the tree. Notice that each node v in the tree with distance $d(v)$ from the root node represents a scheduling policy for the reduced system $\{F_1, \ldots, F_{d(v)}\}$ of minimal forbidden sets (for preselective policies, this equals a partial selection $s = (s_1, \ldots, s_{d(v)})$). Moreover, notice that within the enumeration of linear preselective policies, a node v is deleted from the tree if the partial selection s induced by v cannot be completed to a linear preselective selection. Recall that this can be tested in $O(n + m + f)$ time by checking whether the digraph D of waiting conditions which result from G_0 and s is acyclic, see Section 5.5. According to the above definition of the branching process, for (linear) preselective policies, the number of children of a node v is linear in the number of jobs contained in the associated minimal forbidden set whereas the number of children is quadratic for ES-policies. Moreover, since the number of minimal forbidden sets may be exponential in the number of jobs, the depth of the search tree may be exponential in n for (linear) preselective policies. However, for ES-policies, the maximal depth can be bounded by $n(n-1)/2$ (see Section 6.3.2 below). Figure 6.2 shows the complete search tree of Example 1.3.1 for the case that preselective policies are enumerated. Each node v of the tree contains a letter which indicates whether the partial selection represented by v is linear preselective. For instance, the left most leaf of the tree represents the (linear preselective) selection $(1,2,2)$. (The line style of the nodes refers to domination properties which are discussed later.)

We next consider the *precedence-tree branching scheme* which is described in, e. g., Patterson, Słowiński, Talbot, and Węglarz (1989). Here, linear extensions of the partially ordered set E_0 are enumerated. Each node in the search tree defines an ordering L of some ideal of G_0, that is, $j \in L$ implies that $i \in L$ and $i \prec_L j$

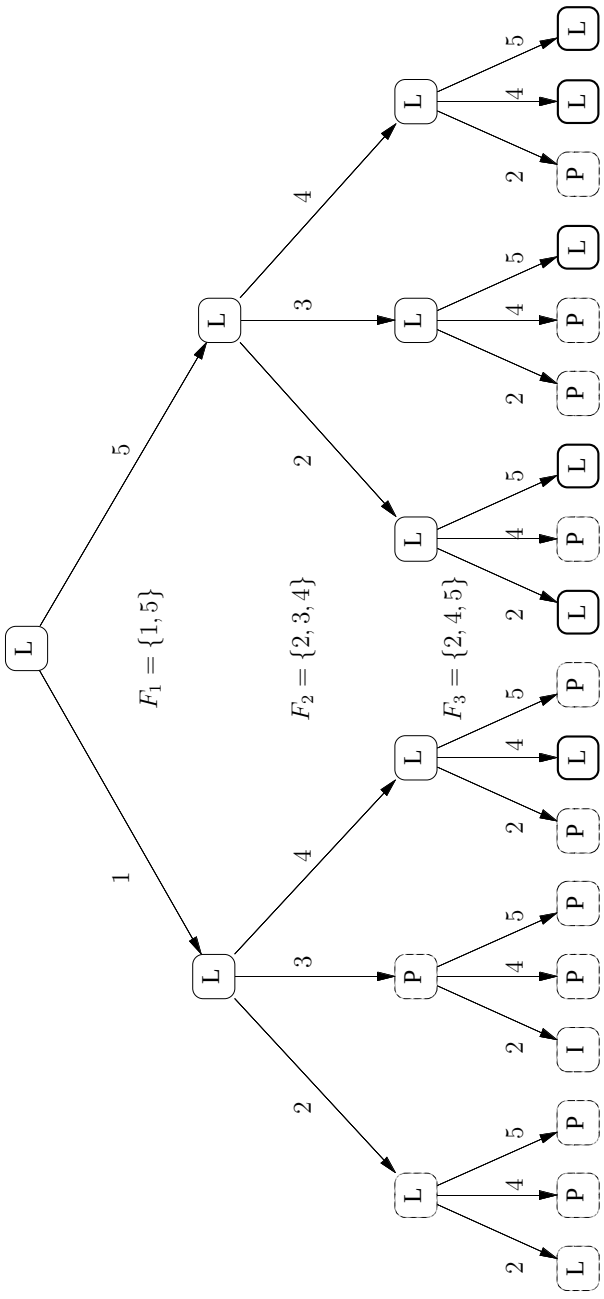

Figure 6.2: The figure shows the complete enumeration tree for Example 1.3.1 based on the forbidden set branching scheme in the preselective case. Arc labels define the preselected job that is chosen to resolve a minimal forbidden set. A node label 'L' indicates that the node represents a linear preselective selection (or a partial selection that can be extended to a linear preselective selection). The node labeled 'I' is infeasible. Nodes with a dashed line style are pruned from the tree because they are dominated according to the algorithm outlined in Section 6.3.2.

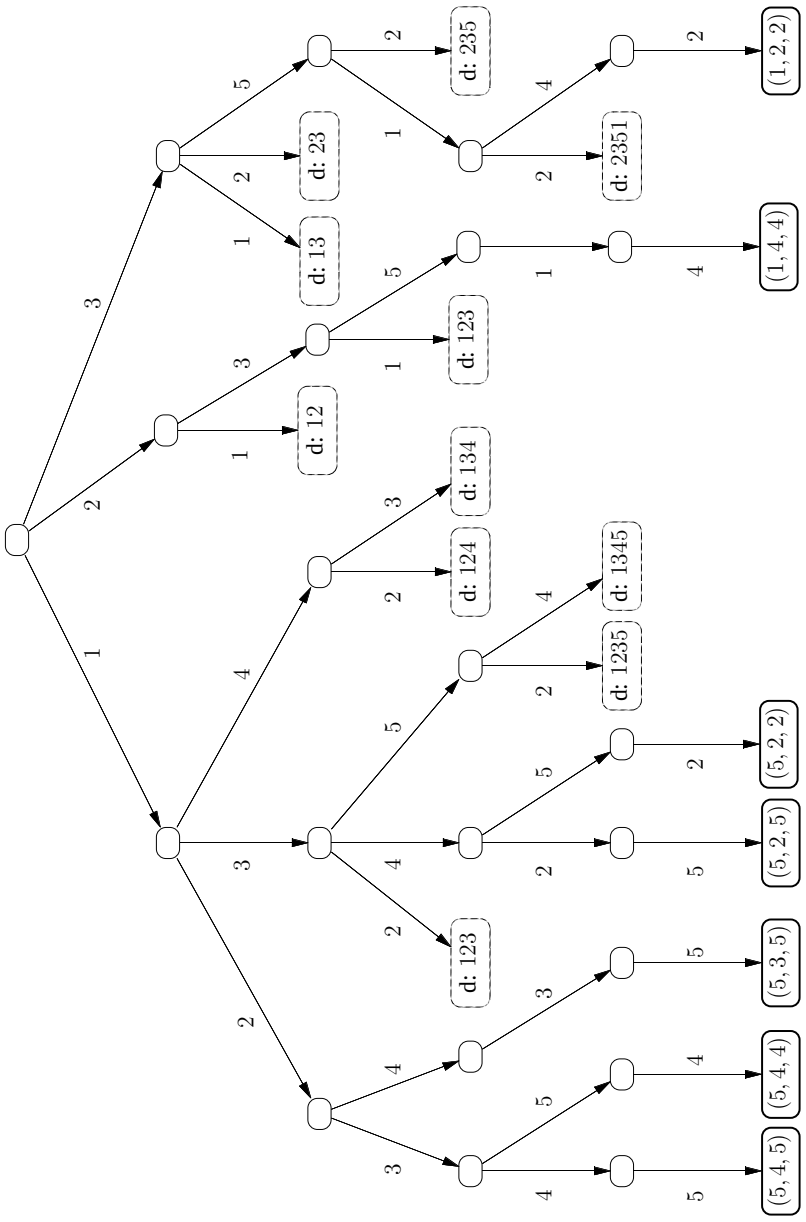

Figure 6.3: The figure shows a part of the enumeration tree for Example 1.3.1 based on the precedence tree branching scheme in the linear preselective case. Arc labels define the job that is appended to the initial segment associated with the parent node. Nodes with a dashed line style are pruned from the tree because they are dominated according to the algorithm outlined in Section 6.3.4. In this case, the node label indicates the ordering which dominates the ordering associated with the node. For instance, the ordering $L = 3 \prec 5 \prec 1 \prec 2$ with node label 'd: 2351' is dominated by $L' = 2 \prec 3 \prec 5 \prec 1$. The node labels with solid line style indicate the associated selection.

for all $(i,j) \in E_0$. We call such orderings L *initial segments* of G_0 (notice that we sometimes view L as a *set* of jobs). For a given node v and associated initial segment L we create child nodes for all jobs $j \in V \setminus L$ that are precedence-feasible, i. e., all predecessors of j are already contained in L. Hence, the set of leaves of the complete tree and the set of linear extensions of E_0 coincide. Notice that the depth of the tree is n, the number of jobs, however, the number of children that are generated for a given node is also in the order of n. Figure 6.3 shows a part of the search tree of Example 1.3.1. For instance, the right most leaf represents the ordering $L = 3 \prec 5 \prec 1 \prec 4 \prec 2$.

So far, the precedence-tree branching scheme has not been applied to stochastic resource-constrained project scheduling problems. A recent reference of its application in the deterministic case is (Sprecher 2000).

Computing the expected makespan. For a given node in the search tree, lower bounds on the expected makespan are computed in order to potentially delete the node from the tree. (A node can be discarded if the computed lower bound is greater than or equal to the current global upper bound.) The lower bound is computed by disregarding all resource conflicts that have not been resolved so far, i. e., we make use of the classical *critical path lower bound* CP. Unfortunately, already for precedence-constrained jobs without any resource restrictions, the computation of the expected makespan is #P-complete if each job processing time distribution has two values (see Section 1.2). We therefore only compute an approximation of the expected makespan with the help of simulation techniques (we noted the basic methodology of simulation in Section 1.2). We then obtain a set P of scenarios for $\boldsymbol{p}$, that can be used to approximately calculate $E[CP(\boldsymbol{p})]$, the expected critical path lower bound for a given node in the search tree. This is done by computing the average critical path lower bound over all scenarios $p \in P$, i. e., $E[CP(\boldsymbol{p})] \approx \frac{1}{|P|} \sum_{p \in P} CP(p)$. As a consequence, we cannot guarantee that we compute an optimal policy for the original input data but instead only for processing time distribution that is induced by the set of scenarios. However, it is known that for $|P| \to \infty$ the optimum value that results from the simulated data converges to the optimum value of the original instance with probability 1. Among other topics, this is proven in (Kleywegt and Shapiro 1999).

6.3 Dominance Rules

In this section we describe five so-called *dominance rules* (one for each branch-and-bound algorithm) that often help to prune large portions of the search trees. For a given node v of some search tree denote by τ_v the set of policies that are defined by the leaves of the subtree rooted at v. We call another node $u \neq v$ *dom-*

inated by v if for each policy $\Pi \in \tau_u$ there exists another policy $\Pi' \in \tau_v$ such that Π' dominates Π (recall the definition from the previous chapter). A *dominance rule* is a method that (usually heuristically) identifies dominated nodes. If such a node is detected, clearly, we can disregard u from further consideration. Notice that we must take care of what might be called *cross pruning*: Suppose that u and v dominate each other; then, pruning both u and v could possibly cut off all optimal solutions.

6.3.1 Earliest Start Policies

We begin the discussion with the class of ES-policies. Let v be a node in the search tree that is constructed by the forbidden set branching scheme. Then v represents an extension $G = (V, E)$ of G_0 that is defined by the pairs that have been chosen on the ancestors of v in order to resolve some of the minimal forbidden sets. Denote by F the minimal forbidden set which is considered for branching at v and suppose that for some pair $(i, j) \in E$ we have $i \in F$ and $j \in F$. Then it is obvious that the child node u_{ij} of v where (i, j) is chosen to resolve F dominates all other child nodes of v. This essentially follows from Lemma 5.3.2. Consider Example 1.3.1 and suppose that the minimal forbidden set $F_1 = \{1, 5\}$ has been resolved by the pair $(5, 1)$. Then, if we next branch on the minimal forbidden set $F_2 = \{2, 3, 4\}$, the branch where $(3, 4)$ is chosen to resolve F_2 dominates all other alternatives.

With respect to an implementation of the above outlined dominance rule, Radermacher (1985) and Bartusch (1984) suggest to use a data structure which they call *destruction matrix*. A destruction matrix stores for each pair (i, j), $i \neq j$, a Boolean array of size f that indicates for each minimal forbidden set F whether $\{i, j\} \subseteq F$. However, this data structure requires $O(n^2 f)$ space, which may easily exceed memory limitations is f is large. We therefore implemented an appropriate algorithm which does not need the memory-expensive destruction matrix. The algorithm was originally developed for preselective policies and is described in the following Section 6.3.2.

6.3.2 Preselective Policies

If preselective policies are enumerated by the forbidden set branching scheme, a node v of the search tree is associated with a partial selection $s = (s_1, \ldots, s_{d(v)})$ ($d(v)$ was defined as the distance of v from the root of the tree). Recall that together with the initial set E_0 of precedence constraints, s defines a set $\mathcal{W}$ of waiting conditions. We know from Chapter 2 and Section 5.4.2 that $\mathcal{W}$ may implicitly resolve other minimal forbidden sets that have not been considered for

Algorithm 9: Dominance rule within the forbidden set based branch-and-bound algorithms

Input : A set V of jobs, a set $\mathcal{F}$ of minimal forbidden sets, and a set of waiting conditions $\mathcal{W}$ induced by E_0 and $s = (s_1, \ldots, s_{d(v)})$.

Output: An extended selection s and the smallest index of a minimal forbidden set that is not resolved by $\mathcal{W}$.

```
for ℓ ∈ {d(v) + 1, ..., f} do
    call Algorithm 1 with input V \ F_ℓ and W_{V\F_ℓ} and
        compute a(j) for each j ∈ F;
    if a(j) > 0 for some j ∈ F then
1       s_ℓ := j;   // or: s_ℓ := −1; (to indicate that F_ℓ is ignored)
    else return (s, ℓ);
return (s, f + 1);
```

branching so far. In particular, following the result of Corollary 5.4.6, we may apply Algorithm 1 to decide whether the minimal forbidden set F which is considered for branching at v is implicitly resolved or not. If F is implicitly resolved then $\mathcal{W}$ implies a waiting condition $(F \setminus \{j\}, j)$ for some $j \in V$. It is a direct consequence of Theorem 5.4.3 that the branch of v where j is the preselected job for F dominates all other child nodes of v.

We have implemented the following variation of the dominance criterion which is applied in each node v of the search tree. In the initially fixed order of the minimal forbidden sets, we check for each currently not resolved minimal forbidden set F, whether F is implicitly resolved by $\mathcal{W}$. If this is the case, we appropriately label F in s and proceed with the next minimal forbidden set. If F is not implicitly resolved we stop the dominance test. For each minimal forbidden set F this test takes $O(n + m + f)$ time. Branching is then performed on the next minimal forbidden set that is neither explicitly resolved by a previous branching, nor labeled to be implicitly resolved. An implementation of the dominance rule is given by Algorithm 9.

Recall the full branch-and-bound tree of Example 1.3.1 as depicted in Figure 6.2. All nodes with a dashed line style are deleted from the tree if Algorithm 9 is included into the branch-and-bound procedure.

The test can equivalently be applied to ES-policies since each partially ordered set $G = (V, E)$ representing an ES-policy can also be expressed as a set $\mathcal{W}$ of waiting conditions. There may be at most $n(n-1)/2$ pairs (i, j) in E, hence the number of OR-nodes required to represent E is $|E|$. Thus the running time of the dominance test reduces to $O(n + |E|)$. In addition, if the dominance test is employed, the maximal depth of the search tree is bounded by $n(n-1)/2$.

6.3.3 Linear Preselective Policies via the Forbidden Set Enumeration

The dominance rule as described in the previous section also applies to the enumeration of linear preselective policies via the forbidden set branching scheme. Consequently, we may use the same algorithm to prune the search tree in this case. However, this causes a problem which requires some explanation. Let the minimal forbidden set F_ℓ be implicitly resolved by the waiting condition $(F_\ell \setminus \{j\}, j)$ with respect to a given partial selection $s^{\ell-1}$. Then, a preselective policy which is induced by extending $s^{\ell-1}$ to a selection s with $s_\ell = j$ is not necessarily linear preselective (we discussed this topic in Section 5.5.3). Hence, the resulting digraph of waiting conditions is not acyclic in general. This causes different drawbacks, in particular, we may no longer use Algorithm 7 to compute earliest job start times; instead, we have to use Algorithm 3 which is of larger worst case complexity. We overcome this difficulty as follows. Instead of choosing j as the preselected job for F_ℓ we simply ignore F_ℓ in all nodes of the search tree that are located in the subtree rooted at the node which induces $s^{\ell-1}$. This is done by assigning an appropriate flag to s_ℓ (instead of $s_\ell := j$), as indicated in the comment of Line 1 of Algorithm 9. Consequently, at the leaves of the search tree, all minimal forbidden sets are resolved and the induced selection yields an acyclic digraph of waiting conditions.

However, with this feature the branch-and-bound algorithm may output a policy that has a smaller optimum value than all linear preselective policies. For instance, if the dominance rule is enabled, the algorithm outputs a minimum makespan of 23.5 for Example 5.7.1 while $\rho^{\mathrm{LIN}} = 23.75$. Now recall the definition of an acyclic preselective policy from Section 5.5.3 and observe that the algorithm always outputs a feasible acyclic preselective policy Π^*; in particular, $\rho^{\mathrm{ACY}} \leqslant \kappa(C^{\Pi^*}(\boldsymbol{p})) \leqslant \rho^{\mathrm{LIN}}$.

Let us briefly account for our decision to employ the dominance rule as outlined above by a discussion of alternatives. First, one may implement a branch-and-bound algorithm that is based on acyclic preselective policies. This, however, allows a lot more freedom of choice because the waiting condition to resolve some minimal forbidden set F is not uniquely determined by the preselected job $j \in F$. The alternative choices for the predecessor set yields additional branches which certainly leads to enormous computation times. In fact, for almost all instances we considered, we obtain $\rho^{\mathrm{PRS}} = \rho^{\mathrm{LIN}}$ which directly yields $\rho^{\mathrm{PRS}} = \rho^{\mathrm{ACY}} = \rho^{\mathrm{LIN}}$. Second, suppose that we only ignore a minimal forbidden set if we know that it is implicitly resolved by a waiting condition that does not induce a cycle in the resulting graph of waiting conditions. Then the computed policy is linear preselective. However, this additional test results into a considerable computational overhead and, if no such waiting job exists, we have to branch over a minimal forbidden set from that we know that the associated resource conflict will not occur.

This makes no sense from a practical point of view. In fact, for all instances we considered, the use of the dominance criterion as formulated in Section 6.3.2 does not yield an optimum expected makespan that is smaller than the value obtained without the dominance rule.

6.3.4 Linear Preselective Policies via the Precedence-Tree Enumeration

In this section and the following Section 6.3.5 we present dominance rules for the branch-and-bound algorithms that are based on the precedence-tree branching scheme. Each node in the precedence tree is identified with an initial segment L of G_0. According to the definition of linear preselective policies we construct a (partial) selection s from L by fixing $s_\ell := j \in F_\ell$ if j is uniquely determined to be the last element of F_ℓ in every initial segment that is obtained if L is extended by the jobs $V \setminus L$. This is clearly the case if at least $|F_\ell| - 1$ jobs from F_ℓ are contained in L.

The dominance criterion is based on the observation that different initial segments L and L' of the same ideal may define identical (partial) selections s. It is then easy to see that, if L and L' are extended by the same job (or by the same ordering of jobs) they still yield the same (partial) selection $\bar{s}$. Clearly, $\bar{s}$ also extends s in the sense that each minimal forbidden set that has been resolved by s is also resolved by $\bar{s}$ and the corresponding preselected jobs coincide. We call an ordering L *dominated* by some other ordering $L' \neq L$ if the policy Π defined by L is dominated by the policy Π' defined by L'. Moreover, denote by $i \prec_{id} j$ that job i has a smaller numbering (or identifier) than job j (we assume without loss of generality that $(i, j) \in E_0$ implies $i \prec_{id} j$).

In the sequel we describe a simple algorithm with the purpose to construct for a given ordering L a lexicographically smaller ordering $L' <_{lex} L$ such that both orderings define the same selection. Then, whenever such an ordering L' exists we delete L from the search tree because it is dominated by L'. Notice that the effect of cross pruning does not occur in this case since for each selection s the lexicographically smallest ordering that induces s is not deleted. Consider now some job i in the ordering L. We traverse L backwards until a predecessor $j \in$ *Pred*$_i$ or some job j with $j \succ_{id} i$ is found ($Pred_i$ denotes the set of predecessors of job i with respect to G_0). Notice that $j \succ_{id} i$ implies that $j \notin Pred_i$ and $j \prec_L i$ implies that $i \notin Pred_j$, hence i and j are unrelated with respect to the precedence constraints. In the first case ($j \in Pred_i$) we stop the dominance test. Otherwise, we check for all minimal forbidden sets F_ℓ with $s_\ell = i$ whether $F_\ell \cap B = \emptyset$. Here, B is defined as the set of jobs h with $i \succ_L h \succeq_L j$. If this is not the case we stop the dominance test. Otherwise we delete L from the search process, since the ordering L' which is obtained from L by moving i to the position directly before

j is lexicographically smaller than L and yields the same selection.

Lemma 6.3.1. *Let s be a selection and let L^* be the lexicographically smallest ordering which represents s. The above outlined algorithm deletes all orderings $L >_{lex} L^*$ from the search tree that yield selection s.*

Proof. Suppose that $L >_{lex} L^*$ is not deleted within the course of the above outlined algorithm and let j and i be the jobs at the left-most position in L and L^*, respectively, with $j \neq i$. It follows that $i \prec_{id} j$, $i \succ_L j$ and that $Pred_i \cap B = \emptyset$, where B is the set of jobs $i \succ_L h \succeq_L j$. Moreover, since L and L^* define the same selection s there exists no minimal forbidden set F_ℓ with $s_\ell = i$ and $F_\ell \cap B \neq \emptyset$. Let $h \in B$ be the last job in L with $i \prec_{id} h$ (h exists because j fulfills the required properties). Denote by L' the ordering which is obtained from L by moving i to the position directly before h. Then $L' <_{lex} L$ and L' yields the selection s. Thus, L is deleted within the course of the above outlined algorithm — a contradiction. □

We employ the above dominance rule for each generated node v of the search tree. The job i in the above description is the last job in the initial segment L defined by v (it is not hard to see that the above argumentation is also valid for incomplete orderings $L \subset V$). The dominance rule can be implemented to run in $O(nf)$ time. Recall the branch-and-bound tree of Example 1.3.1 as depicted in Figure 6.2. All nodes with a dashed line style (and also the not shown subtrees rooted at these nodes) are deleted from the tree if the dominance rule is included into the branch-and-bound procedure.

6.3.5 Job-Based Priority Policies

Similar to linear preselective policies, for a given ordering L of jobs, we aim at constructing another ordering L' by moving a single job i to another position such that L' dominates L. Notice that we assume in this section that a given ordering is evaluated according to Algorithm 8. Sprecher (2000) reports on various dominance rules for the case of deterministic processing times, however, each of the rules described there requires fixed job processing times and is thus not applicable in the stochastic case. Instead, we make use of the following, simple dominance criterion. For a given ordering L and $i \in L$ we denote by L_i the ordering induced by L on the elements $\{h \in L | h \preceq i\}$.

Lemma 6.3.2. *Let L be an ordering of the jobs and let i and j be jobs with the properties $j \prec_L i$ and $Pred_i \subseteq Pred_j$. Denote by L' the ordering obtained from L by moving i directly to the position before j. Moreover, let $B = L_i \setminus \{h \in L_i | \forall p \in \mathbb{R}^n_> \text{ holds } C_h(p) \leqslant S_j(p)\}$. If i is contained in no minimal forbidden set F with $F \subseteq B$ then L is dominated by L'.*

Proof. Since L is an extension of G_0 and $Pred_i \subseteq Pred_j$ we have that i is not related to any job h with $i \succ_L h \succeq_L j$. Consequently, L' is an extension of G_0. For an arbitrary scenario $p \in \mathbb{R}^n_>$ let $S(p)$ and $S'(p)$ denote the schedules resulting from Algorithm 8 with input p and L and L', respectively. In the order of L' we show for each job h that $S'_h(p) \leqslant S_h(p)$. For $h \prec_{L'} i$ this is trivial, since $S'_h(p) = S_h(p)$. For i we obtain $S'_i(p) \leqslant S_j(p) \leqslant S_i(p)$. To see this, we show that $t := S_j$ is a feasible start time for i after the jobs $h \prec_{L'} i$ have been scheduled by Algorithm 8. Since $Pred_i \subseteq Pred_j$, starting i at t is time-feasible. Consider now the set B' of jobs that are in process at t. It follows from the feasibility of $S(p)$ that $B' \cup \{j\}$ is resource-feasible and that $B' \cup \{j\} \subseteq B$. Since i is contained in no minimal forbidden set F with $F \subseteq B$, starting i at t is resource-feasible and thus feasible. As a consequence, $S'_i(p) \leqslant S_j(p)$; $S_j(p) \leqslant S_i(p)$ follows from the definition of Algorithm 8. By essentially the same argumentation we have that $S'_h(p) = S_h(p)$ for all $i \succ_L h \succeq_L j$. Finally, recall that the orderings L and L' on the remaining jobs $h \succ_L i$ coincide, hence $S'_h(p) \leqslant S_h(p)$ is also valid for these jobs. □

A dominance rule based on Lemma 6.3.2 may now be implemented as follows. For each node in the search tree with associated initial segment L of some jobs we let job i (as in the lemma) be the last job in L. Notice that L needs not contain all jobs of V. Job j is identified by traversing L backwards starting from i until j fulfills $Pred_i \subseteq Pred_j$ (or $j \in Pred_i$ in which case the dominance test is aborted). Then, we compute B (as defined in Lemma 6.3.2) and test whether there is a minimal forbidden set F with $i \in F$ and $F \subseteq B$. If there is no such set, we know that L is dominated and hence we delete the associated node in the search tree.

However, there are some difficulties that should be discussed. First, recall that the reason why we consider job-based priority policies is to avoid the handling of exponentially many minimal forbidden sets. We have the threshold-representation of resource constraints only and consequently, we lose some structural information which we can access within the computation of optimum (linear) preselective policies and ES-polices. In particular, as we have shown in Theorem 4.2.3 we cannot decide efficiently whether a given job i is contained in some minimal forbidden set or not, unless P=NP. However, recall from the discussion in Section 4.2.2 that it can be answered in polynomial time whether i is contained in some forbidden, but not necessarily minimal forbidden set together with the jobs in B. We employ this algorithm and if i is not contained in a forbidden set F with $F \subseteq B$ then it is not contained in a minimal such forbidden set. Hence we know that L is dominated. Contrarily, if i is contained in a forbidden set $F \subseteq B$, we stop the dominance test. Another topic that has to be addressed is cross pruning. As an example, consider an instance with $V = \{1, 2\}$, $E_0 = \emptyset$, and $\mathcal{F} = \emptyset$. Then $1 \prec 2$ and $2 \prec 1$ are the resulting linear extensions which clearly dominate each

other. To handle the difficulty of cross pruning we only discarded an ordering from the search process if the involved jobs i and j (as in Lemma 6.3.2) fulfill $j \succ_{id} i$. Then, among a set of orderings that pairwise dominate each other, the lexicographically smallest ordering is not discarded from the search process.

6.4 Improving the Performance

As usual for branch-and-bound procedures we make use of several additional features in order to speed up the computations. In the sequel we briefly explain how we compute initial upper bounds, several lower bounds, a flexible tree traversal strategy as well as in which order minimal forbidden sets should be considered for branching. The computational impact of these ingredients are presented in Section 6.5.4 below.

6.4.1 Initial Upper Bound

In order to obtain an initial valid upper bound to start with, we use list scheduling algorithms applied to ten standard priority rules, such as, e. g., *shortest/longest processing time first* and *minimum slack*, see, e. g., (Kolisch 1996) for details. They clearly require deterministic processing times; we have chosen the expected job processing times $E[\boldsymbol{p}]$. From the resulting schedules we choose one schedule S with minimum makespan. Notice that we may additionally choose any other heuristic for the deterministic resource-constrained project scheduling problem to generate a feasible schedule (see Section 1.1). The ordering L of jobs obtained from non-decreasing start times S_j clearly respects the precedence constraints, hence, L directly yields a linear preselective and a job-based priority policy. Their expected makespans are taken as initial upper bounds, respectively. Moreover, for the preselective approach we use the linear preselective upper bound as well. Finally, an earliest start policy is constructed by choosing for each minimal forbidden set a pair (i, j), $i, j \in F$ in such a way that i has minimum completion time in S and j has maximum start time in S. Its expected makespan is taken as initial bound for ES-policies.

6.4.2 The Critical Path Lower Bound and Jensen's Inequality

Recall that the computation of the critical path lower bound is time consuming in the stochastic case because we have to compute the deterministic equivalent for each scenario $p \in P$. As a consequence, we additionally employ a variation of this bound which can be computed more efficiently. The bound is based on Jensen's inequality and is applied before the expensive simulation-based compu-

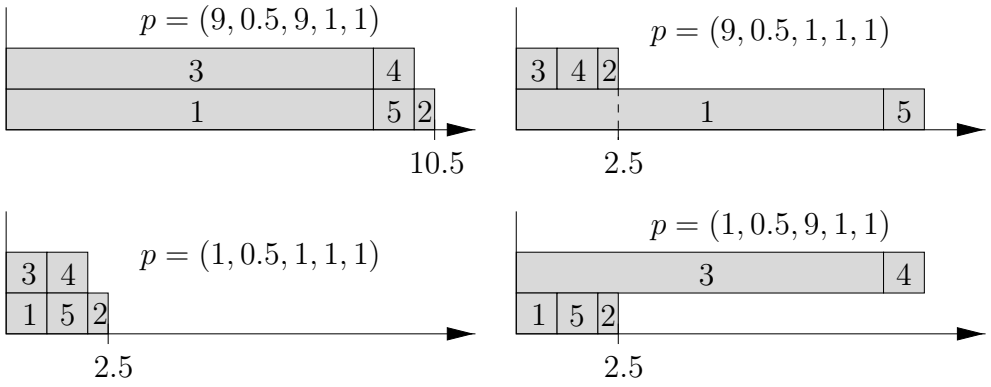

Figure 6.4: The figure shows a Gantt chart for each of the four possible scenarios resulting from Example 6.4.1.

tation. Then, if it is greater than or equal to the current global upper bound, the expected critical path length needs not to be computed.

In the case of *ES-policies* we can simply calculate the deterministic critical path lower bound with respect to $E[\boldsymbol{p}]$ which is a lower bound for each convex cost function, hence also for C_{max}. This is immediate with Jensen's inequality and due to the fact that the earliest start computation is a convex function of the job processing times. However, the computation of job start times is not convex for (linear) preselective policies and job-based priority policies (since jobs can be started at the *minimum* of completion times of other jobs). In fact, for a given minimal forbidden set F, the expected completion time of a preselected job may even be less than the minimum of the expected start times of the other jobs in F. The following example illustrates this effect.

Example 6.4.1. *Let $G_0 = (V, E_0)$ be given by $V := \{1, 2, 3, 4, 5\}$ and $E_0 := \{(1, 4), (3, 5)\}$ and let the set $F = \{2, 4, 5\}$ be minimal forbidden. The following processing times are deterministic: $p_2 = 0.5, p_4 = 1, p_5 = 1$. The processing times of jobs 1 and 3 are independently distributed with $Pr(p_1 = 1) = Pr(p_1 = 9) = \frac{1}{2}$ and $Pr(p_3 = 1) = Pr(p_3 = 9) = \frac{1}{2}$. Furthermore, let job 2 be the preselected job in the minimal forbidden set F.*

For the deterministic problem with expected job processing times we have $S_2(E[\boldsymbol{p}]) = 6$ while $E[S_2(\boldsymbol{p})] = 4$ (see Figure 6.4). Note that even the expected *completion* time $E[C_2(\boldsymbol{p})] = 4.5$ is less than the minimum of the expected *start* times of the other jobs in the minimal forbidden set ($E[S_4(\boldsymbol{p})] = E[S_5(\boldsymbol{p})] = 5$).

We handle this effect as follows. Whenever a minimum (of random variables) has to be computed, we make use of the component-wise smallest processing times p^{min} of the set of generated scenarios p. As for ES-policies, each maximum computation is bounded by Jensen's inequality. We adapt (5.2) by setting $S'_a := 0$

and

$$S'_j := \max\{\max_{\substack{(X,j)\in\mathcal{W},\\ |X|\geqslant 2}}(\min_{i\in X}(S'_i + p_i^{min})), \max_{(\{i\},j)\in\mathcal{W}}(S'_i + E[\boldsymbol{p}_i])\}$$

for all $j \in V$.

Lemma 6.4.2. *$S'_j \leqslant E[S_j(\boldsymbol{p})]$ for all jobs $j \in V$.*

The lemma follows directly from Jensen's inequality and the monotonicity of preselective policies. As a consequence, the start time S'_b of the dummy job b which indicates the project completion, is a lower bound for $E[C_{max}(\boldsymbol{p})]$ for both preselective and linear preselective policies. By essentially the same technique we can also derive a lower bound for job-based priority policies.

Finally, notice that different, more sophisticated procedures have been devised in the literature in order to compute lower and/or upper bounds on the expected makespan when jobs are only precedence-constrained (*PERT-networks*). We made experiments with an adaption of the approach as proposed by Devroye (1979) (see also (Arnold 1988)), however, the results are of the same order of magnitude as the lower bound based on Lemma 6.4.2. We therefore did not include such bounds into our experiments.

6.4.3 Single Machine Scheduling Relaxations

Besides the above mentioned critical path based lower bound we employ a well known lower bound which is based on a single machine relaxation of the original problem. Variations of this bound are frequently used in deterministic project scheduling, see, e. g., (Mingozzi, Maniezzo, Ricciardelli, and Bianco 1998; Sprecher 2000). For a given node in the search tree let $head_j$ be a lower bound on the expected start time $E[S_j(\boldsymbol{p})]$ of job j. Moreover, let $tail_j$ be a lower bound on the expected makespan of the subproject that is induced by the successors of j in G_0. We obtain the following lemma.

Lemma 6.4.3. *Let $W \subseteq V$ be a subset of jobs that can pairwise not be scheduled in parallel. Then $\min_{j\in W}(head_j) + \sum_{j\in W} E[\boldsymbol{p}_j] + \min_{j\in W}(tail_j)$ is a lower bound on the expected makespan for all preselective policies.*

Proof. Each preselective policy plans the jobs of W in a fixed order, independently of the job processing times. Let i and j be the first and the last job in that order, respectively. Then, $head_i + \sum_{h\in W} E[\boldsymbol{p}_h] + tail_j$ is a lower bound on the expected makespan. Since i and j are unknown we choose the smallest possible values for the expected start of i and the tail for j. □

Since the bound is valid for preselective policies it also holds for linear preselective, job-based, and ES-policies. Notice that the bound is not valid for arbitrary policies. If the order in which the jobs of W are scheduled is dependent on the processing times of their predecessors, it is easy to construct a counter example: Consider the jobs $V = \{1, 2, 3, 4\}$ with $E_0 = \{(1, 3), (2, 4)\}$, $p_3 = p_4 = 3$, and $p_1 = p_2 \in \{1, 5\}$ (each with probability $\frac{1}{2}$). Suppose that the jobs $W = \{3, 4\}$ form a minimal forbidden set, i. e., they must be scheduled sequentially. Then the formula of Lemma 6.4.3 yields a 'lower bound' of 9 while the expected makespan of any (resource-based) priority policy is 8.5.

We have included the following implementation into our experiments. As a preprocessing step we compute $tail_j$, $j \in V$, as the expected makespan of the subproject that is induced by the successors of j (resource constraints are ignored). Moreover, we compute different sets W by simple priority-rule heuristics: we start with $W = \emptyset$, and consider the jobs in the order defined by the priorities and add job j to W if j cannot be processed simultaneously with any job that has been previously added to W. This takes $O(n^2)$ time per computed set. Then, for each node which is explored in the search tree we compute a lower bound on the expected start time of the jobs according to Lemma 6.4.2, i. e., we set $head_j := S'_j$. The resulting single machine instance with heads and tails is fed into an algorithm as proposed by Carlier (1982, Proposition 1). Carlier's algorithm uses the fact that for given set W of jobs, depending on the heads and tails of jobs, a subset $W' \subset W$ may result in a better lower bound. The algorithm computes the best lower bound that can be achieved from any subset of W by a preemptive relaxation in $O(|W| \log |W|)$ time. Finally, we choose the maximum of the computed bounds over all sets W which is a lower bound on the expected makespan of the project.

6.4.4 Sorting the Minimal Forbidden Sets

An important ingredient of branch-and-bound algorithms in general is to find an appropriate ordering of the decisions that have to be made. It is of great advantage to perform those branchings early that lead to a large increase of the overall lower bound, i. e., the gap between lower and upper bound is reduced as early and as much as possible. Furthermore, it often pays off to first perform such branchings where only few alternatives have to be explored.

The forbidden set branching scheme easily allows to exploit these general ideas: before starting the full branch-and-bound algorithm we explore for each minimal forbidden set F the alternative where F is selected for the first branching. For each such alternative we compute the set B of branches that cannot be discarded because their lower bound is less than the initial global upper bound. In the full branch-and-bound algorithm we then resolve the minimal forbidden sets in

the order of increasing $|B|$. Notice that, if none of the branches can be pruned by lower bound computation, the minimal forbidden sets are ordered by increasing cardinality. As a tie-breaker we choose the average increase of the lower bound taken over all branches that result from a single minimal forbidden set. More precisely, if LB_b is the lower bound of a branch $b \in B$ then the sorting key that we have chosen is $\sum_{b \in B} LB_b$.

6.4.5 Flexible Search Strategy

As another standard trick for branch-and-bound, we implemented a flexible tree traversing strategy that simultaneously processes a parameter driven number of DFS-like paths at a time. In contrast to simple backtracking procedures, such search strategies usually do not waste too much time in useless parts of the search tree. Moreover, in order to decide which node is chosen next for branching, we assign a priority to each of the nodes in the tree. The priority is computed as a combination of the lower bound on the expected makespan and the depth of that node in the search tree.

6.5 Computational Study

The computational study is divided into five parts. In the first two parts we describe the computational setup and the considered test set. We then study the performance of each of the five branch-and-bound procedures relative to each other and also in dependence from instance characteristics. In the fourth part we analyze the impact of the additional ingredients described in Section 6.4. Finally, we study the behavior of the algorithms on larger instances.

For ease of reference we use the following abbreviations for the five branch-and-bound algorithms. The algorithms to compute optimum preselective policies and ES-policies (both forbidden set branching scheme) are denoted by PRS-FS and ES-FS, respectively. The enumeration of job-based priority policies via the precedence tree branching scheme is abbreviated by JBP-PT and the two variations of computing linear preselective policies (forbidden set and precedence tree branching scheme) are referred to as LIN-FS and LIN-PT, respectively.

6.5.1 Computational Setup

Our experiments were conducted on a Sun Ultra 1 with 143 MHz clock pulse operating under Solaris 2.7. The code has been written in C++ and is compiled with the GNU g++ compiler version 2.91.66 using the -O3 optimization option. We allowed the algorithms to maximally use 50 MB of main memory and a time-

limit of 1000 seconds.

In order to establish a reference setting for the various parameters we have performed different initial experiments. Based on the results of these experiments we decided to set the parameter defaults as follows. The computation of an initial upper bound (Section 6.4.1) as well as the lower bounds described in Sections 6.4.2 and 6.4.3 are switched on. Furthermore we employ the search strategy as described in Section 6.4.5 by considering three DFS-like paths at a time. In each of the branch-and-bound algorithms we have enabled the respected dominance rules as described in Section 6.3. For the algorithms that are based on the forbidden set branching scheme, we also performed the sorting of forbidden sets as introduced in Section 6.4.4. The default type of the distribution of job processing times is a Gamma distribution with a maximum variance of 3. Finally, we generate 200 scenarios from the distributions, which turned out to provide a reasonable tradeoff between the precision of the expected makespan on the one hand and the computational effort on the other hand.

Unless we mention explicitly that some parameter is modified we always report on experiments that are based on the above defined parameter setting. The impact of most of the parameter settings is documented in detail in Section 6.5.4 below.

6.5.2 The Test Sets

We have applied our algorithms to a test set which is created by the instance generator ProGen (Kolisch and Sprecher 1996). The test set contains 480 instances each of which consists of 20 jobs. Each job requires at most 4 different resources and comes with an integral deterministic processing time which has been chosen randomly between 1 and 10. The average number of minimal forbidden sets in this test set is roughly 70 (maximum 774). Equivalently to the test sets described in Section 4.4.1, the instances have been generated by modifying the instance parameters network complexity, resource factor, and resource strength. The parameters have been chosen out of the sets $NC \in \{1.5, 1.8, 2.1\}$, $RF \in \{0.25, 0.5, 0.75, 1.0\}$, and $RS \in \{0.2, 0.5, 0.7, 1.0\}$, respectively. This leads to 48 combinations and for each combination we have created 10 instances. In Section 6.5.5 below we also report on instances with 30 and 60 jobs, respectively (the generation of these test sets was described in Section 4.4.1).

We next explain how we generate the probability distributions of the job processing times (which are not created by the ProGen instance generator). We take the given deterministic processing time of each job as expectation. Then, together with different, parameter driven values for the variance we construct uniform and triangle, as well as approximate normal, Gamma, and exponential distributions.

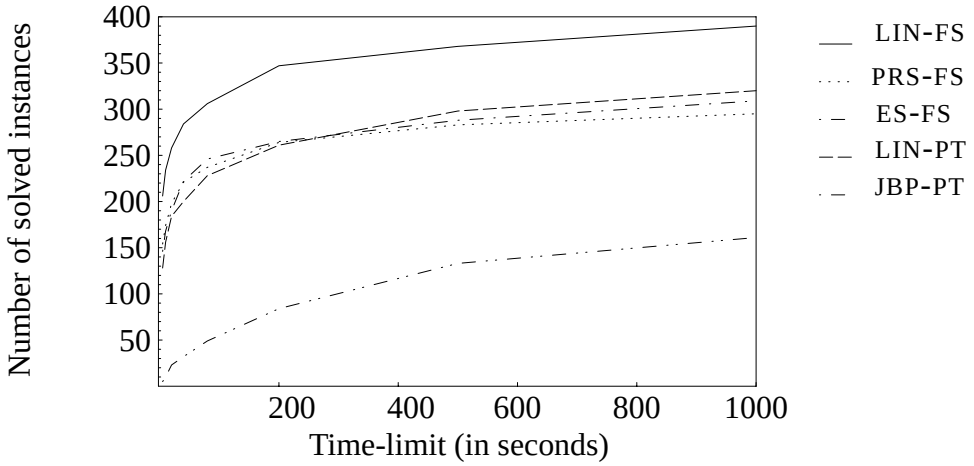

Figure 6.5: The number of optimally solved instances (out of 480 instances with 20 jobs each) depending on given time-limits. The ordering for the time limit of 1000 seconds is as follows (from top to bottom): LIN-FS, LIN-PT, ES-FS, PRS-FS, JBP-PT.

By appropriate rounding we make sure that $Prob(\boldsymbol{p}_j < 0) = 0$. Finally, the scenarios p from $\boldsymbol{p}$ are generated by standard simulation techniques, where job processing times are assumed to be independent.

6.5.3 Comparison of the Procedures

We say that an algorithm A *optimally solves* (or *optimizes*) a given instances (within a given time limit) if A finds an optimal solution and verifies the solution to be optimal. We characterize the *performance* of A by the number of optimally solved instances in the considered test set and the average computation time among the optimally solved instances.

Performance of the different procedures. We start the study by reporting on the computational expenses that are required by the different algorithms. Figure 6.5 shows for each of the five algorithms how many of the 480 instances can be solved optimally for different time limits. The plot indicates that, if linear preselective policies are enumerated by the forbidden set branching scheme, considerably more instances were solved when compared to the other algorithms. The plot also demonstrates that the precedence tree enumeration works quite satisfactory for linear preselective policies; it solved more instances to optimality than preselective policies, ES-policies, and job-based priority policies for time limits greater than 300. It turns out that, for most of the considered instances, LIN-FS

works much faster than LIN-PT, however, roughly 10% of the instances can be solved faster by the precedence tree enumeration than by the forbidden set enumeration. On the shady side we observe that the enumeration of job-based priority policies is extremely time intensive. Only 161 out of 480 instances were solved optimally within a time limit of 1000 seconds. The dominance rule as proposed in Section 6.3.5 is probably too weak and prunes not enough parts of the search tree. In fact, the number of nodes that is evaluated from JBP-PT within the search exceeds the number of nodes of LIN-PT by a factor of 3 on average (among 155 instances solved by both procedures). However, we also observe that there are 12 instances that were optimally solved with JBP-PT but not with LIN-FS.

Finally, it should be noted that, except for ES-FS, the limited memory was not a critical resource (no experiment had to be aborted). For ES-FS, the allocated memory has exceeded the limit of 50 MB for 87 instances. The reason probably is that the number of children created at each branching is $\mathrm{O}(\,n^2\,)$ compared to $\mathrm{O}(\,n\,)$ for the other procedures (see Section 6.2).

Dependency on instance characteristics. Let us next discuss the performance of the algorithms in dependence from different instance characteristics. Instead of considering the parameters NC, RF, and RS, we first concentrate on a much more important characteristic, namely the number $f := |\mathcal{F}|$ of minimal forbidden sets. It is not surprising that f is a dominating parameter for the performance of each of the forbidden set based procedures (PRS-FS, LIN-FS, ES-FS, and also LIN-PT). Figure 6.6 displays the interrelations. The figure consists of six parts, the first of which shows the number of instances in the test set depending on the number of minimal forbidden sets. A peak of height h at position ℓ means that there are h instances in the test set which have ℓ minimal forbidden sets. The other charts (2–6) display the number of instances that were optimally solved by the branch-and-bound algorithms in dependence from the number of minimal forbidden sets in the respective instance (h instances with ℓ minimal forbidden sets were solved optimally). Let us first exclude the procedure JBP-PT from the discussion. If the number of forbidden sets is small (say, less than 50) we observe that (almost) all instances in the test set can be solved optimally by PRS-FS, LIN-FS, ES-FS, and LIN-PT. These procedures also optimize many of the instances with 50–100 minimal forbidden sets, however, we see that LIN-FS performs considerably better than PRS-FS, ES-FS, and LIN-PT. Finally, among the instances which comprise of more than 100 minimal forbidden sets are relatively few that have been optimally solved. With respect to JBP-PT, according to Chart 6 of Figure 6.6, we observe a comparatively small dependency on f. Even among the instances with very few minimal forbidden sets ($f \leqslant 50$) there are quite many which cannot be solved optimally. On the other hand, instances that are optimized by JBP-PT but not by

one of the algorithms PRS-FS, LIN-FS, or ES-FS, contain a comparatively large number of minimal forbidden sets (there exist 12 such instances and their average number of minimal forbidden sets is 157).

We next analyze the performance of the algorithms in dependence from the instance parameters network complexity NC, resource factor RF, and resource strength RS. Again, let us start by discussing the algorithms PRS-FS, LIN-FS, ES-FS, and LIN-PT. As a consequence of the strong impact of the number of minimal forbidden sets, the dependency of the algorithm's performance on NC, RF, and RS, is dominated by the impact of these parameters of the number of minimal forbidden sets: If the combination of NC, RF, and RS yield a small number of minimal forbidden sets the performance is excellent and vice versa. As a consequence, the impact of NC, RF, and RS on the branch-and-bound algorithms can directly be derived from the computational results presented in Chapter 4. With respect to JBP-PT let us discuss the dependency of the performance on the resource strength. We observe that, if RS is small, only very few instances can be solved to optimality and vice versa. The average computation time is 393 seconds among 7 optimally solved instances with $RS = 0.2$ while 190 seconds on average are needed to solve instances with $RS = 1.0$ to optimality (86 instances). The dependency is depicted in Table 6.1, Row 2. In the same table (Row 3) we show the number of instances that were solved optimally by LIN-FS. Finally, in Row 4 we give the average number of minimal forbidden sets subject to RS. We observe the above mentioned correlation between the number of minimal forbidden sets and the performance of LIN-FS. It is maybe an interesting observation that instances with scarce resources, i. e., $RS = 0.2$, seem to be solvable with reasonable effort by the forbidden set based algorithms: For instance, LIN-FS optimally solves 104 out of 120 instances within an average computation time of 114 seconds. Instances with scarce resources (with low resource strength) often appear to be the hardest instances in the PSPLIB. For the deterministic case, this was observed by various authors, e. g., by Dorndorf, Pesch, and Phan Huy (2000a) in the context of a branch-and-bound procedure and by Möhring, Schulz, Stork, and Uetz (2000) in connection to lower bound computations.

Comparison of optimum costs. In extension to Section 5.7 we next discuss how the different values of the optimal expected makespan of the considered classes of policies are related to each other. Figure 6.7 shows the average and the maximum of the optimum expected makespan taken over 107 instances that were solved by all procedures. The values have been scaled such that they represent the percental deviation from the deterministic optimum makespan (with respect to $E[\boldsymbol{p}]$). By definition, preselective policies yield the smallest expected makespan among all considered classes of policies. However, surprisingly perhaps, the other

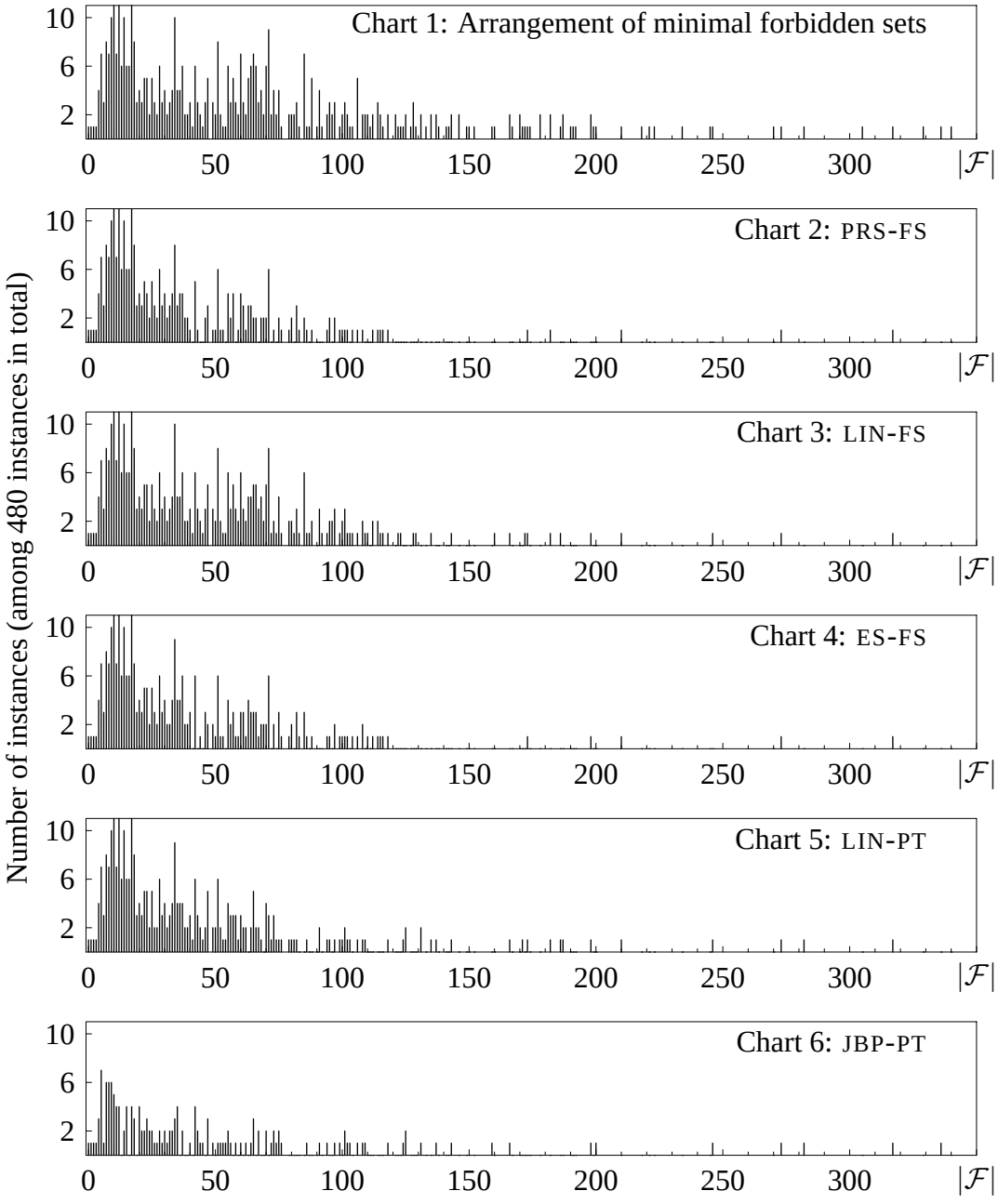

Figure 6.6: The first chart shows the number of instances in the test set in dependence from the number of minimal forbidden sets. The other charts (2–6) display the number of instances that were optimally solved by the branch-and-bound algorithms in dependence from the number of minimal forbidden sets in the respective instance.

Algorithm	$RS = 0.2$	$RS = 0.5$	$RS = 0.7$	$RS = 1.0$
JBP-PT	7	24	44	86
LIN-FS	104	75	98	113
average $\|\mathcal{F}\|$	62	91	78	47

Table 6.1: The table shows the dependency of the number of optimally solved instances subject to the resource strength RS (Rows 2 and 3). In addition, the figures in Row 4, give the average number of minimal forbidden sets for each of the subsets of instances with same resource strength.

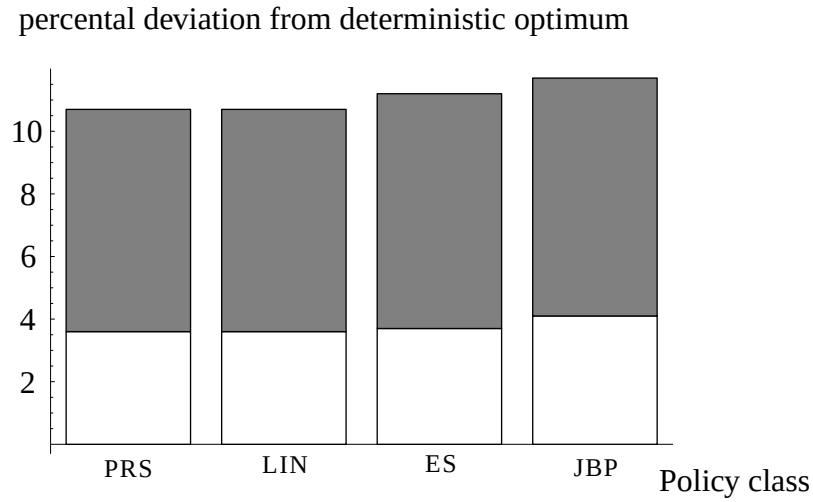

Figure 6.7: Average and maximum percental deviation of the expected makespan from the deterministic optimum. The figures are based upon 107 instances that were solved optimally by all algorithms.

classes of policies yield values that are at most 0.5% worse on average (maximal 2.1%). In particular, preselective policies and linear preselective policies yield exactly the same optimum costs for all but 4 instances (among the 295 instances that can be solved by both PRS-FS and LIN-FS). Furthermore, for our test set, the average optimum value of job-based priority policies is roughly 0.4% worse on average (maximum 1.8%) when compared to ES-policies (recall that these classes are incomparable with respect to the optimum expected makespan).

Notice that Figure 6.7 also exposes that a deterministic planning may yield very optimistic estimates. We see that, on average, the expected makespan is more than 3.5% larger than the deterministic makespan (with respect to processing times $E[\boldsymbol{p}]$). Even more, the maximal percental deviation is occasionally greater than 10%.

6.5.4 Impact of Additional Ingredients

In this section we test how the dominance rules as well as the various ingredients proposed in Section 6.4 help to reduce the computation times.

Impact of dominance rules. We next consider the impact of the dominance rules on the performance of the algorithms. Table 6.2 shows the results for each of the procedures. In the first column we show the used algorithm, the second and the third column refer to the results for the standard parameter setting, i. e., the dominance rules are switched on. The forth and the fifth column document the experiment where no dominance rule is employed. We see that the dominance rules significantly improve the performance of the branch-and-bound procedures. In particular, within the precedence tree enumeration (JBP-PT and LIN-PT), the number of instances that can be solved to optimality when the dominance rules are switched off is reduced to roughly one third. Hence, although the dominance rules used in JBP-PT and LIN-PT are not strong enough to compete with LIN-FS, they cut off quite large portions of the search tree.

Impact of lower bounds. All relevant data concerning the impact of the lower bounds as described in Section 6.4 is displayed in Table 6.3. In the first column we state the used algorithm, the second and the third column documents the results for the standard parameter setting, i. e., the single machine bound and the critical path bound based on Jensen's inequality are enabled. The forth and the fifth column refer to the experiments where the single machine based bound is disabled. Finally, Columns 6 and 7 document the case where both the machine-based bound and the critical path bound based on Jensen's inequality is switched off (recall that the single machine relaxation requires the output of the critical path based bound).

	dominance on		dominance off	
Algorithm	#inst. opt.	∅ CPU	#inst. opt.	∅ CPU
PRS-FS	295	21	235	89
LIN-FS	390	29	348	74
ES-FS	309	9	188	69
LIN-PT	320	3	100	162
JBP-PT	161	70	57	394

Table 6.2: The impact of the dominance rule as described in Section 6.3. The columns '#inst. opt.' denote the number of instances for which an optimal policy was found and proven to be optimal. The columns '∅ CPU' refer to the required computation times in seconds. Notice that computation times are only comparable row by row. Within each row, they are based on the instances that can be optimized by both variations (dominance on or dominance off).

For each variation of the parameters we show the number of solved instances as well as the average computation times in seconds.

The figures indicate that both lower bounds result into improvements with respect to the number of optimally solved instances as well as the associated computation times. Notice that, for the single machine relaxation, one cannot expect exceptional good results on average. The relaxation only considers minimal forbidden sets of cardinality 2, which makes it rather weak for instances with only few such minimal forbidden sets. However, for instances with many such forbidden sets the bound leads to a considerable improvement of computation time, sometimes to more than 50% (21 instances for LIN-PT). The bound that is based on Jensen's inequality leads to remarkable improvement for the case of ES-FS. Here, we do not have to make use of the minimal processing time of jobs p^{min}. Since the lower bound for all other procedure relies on p^{min} (recall Section 6.4.2) its effect on the computation is weaker. However, computation times are reduced considerably.

Impact of the search strategy. We document the impact of the used strategy to traverse the search tree in comparison to a classical depth-first search (DFS) procedure. The results are displayed in Table 6.4. In the first column we show the used algorithm, the second and the third column refer to the results for the standard parameter setting, i. e., the flexible search strategy is employed. The forth and the fifth column document the case where depth-first search is used. Again, for each variation we state the number of solved instances as well as the average computation times in seconds. In all cases the number of instances that can be solved optimally is considerably larger if the search strategy as described in

	std. param. setting		machine LB off		mach./Jensen LB off	
Algorithm	#inst. opt.	∅ CPU	#inst. opt.	∅ CPU	#inst. opt.	∅ CPU
PRS-FS	295	57	291	62	290	73
LIN-FS	390	67	388	69	383	84
ES-FS	309	56	308	59	294	98
LIN-PT	320	90	302	96	297	109
JBP-PT	161	239	161	241	151	268

Table 6.3: The impact of the lower bounds as described in Section 6.4. The average computation times are only comparable row by row. Within each row, they are based on the instances that can be optimized by all variations (all lower bounds on, single machine lower bound off, both single machine lower bound and the bound based on the Jensen inequality off).

	flexible search		depth-first search	
Algorithm	#inst. opt.	∅ CPU	#inst. opt.	∅ CPU
PRS-FS	295	57	277	83
LIN-FS	390	60	372	83
ES-FS	309	69	287	92
LIN-PT	320	83	304	115
JBP-PT	161	151	111	358

Table 6.4: The impact of the search strategy as described in Section 6.4. The average computation times are only comparable row by row. Within each row, they are based on the instances that can be optimized by both variations (flexible search or simple depth first search).

Section 6.4.5 is employed. Even more, the average computation time is drastically smaller when compared to the depth-first search traversal.

Impact of stochastic parameters. We next analyze the impact of the stochastic parameters, that is, the type and the variance of the processing time distributions. Note that we only document the results obtained for LIN-FS, since the behavior of each of the other algorithms is analogous (with respect to the conclusions we draw). For the different types of distributions we observe that the performance of the procedures is not significantly affected. Except for exponential distributions, Algorithm LIN-FS optimizes for all considered types of distributions roughly 390 out of the 480 instances at an average computation time of 70 seconds per instance. For exponential distributions the number of instances solved optimally is only 379 and the required computation time is larger. For the 379 instances the

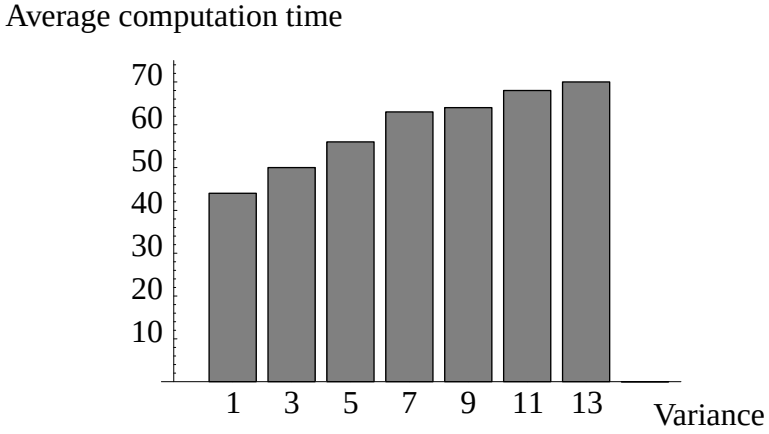

Figure 6.8: Dependency of the computation time (in seconds) on the chosen variance of the distributions (obtained with LIN-FS). The figures are based upon 366 instances that were solved optimally for each variance setting ($\{1, 3, 5, 7, 9, 11, 13\}$) for the random job processing times. The used type of distribution was the Gamma distribution.

algorithm required 77 seconds on average while for other distributions only 55 seconds are required (for these instances). This is probably due to the fact that for strongly varying processing times (which is the case for exponential distribution, since we used a larger support when compared to the other types of distributions) the computation time increases. Figure 6.8 displays the dependency of the computation time on the chosen variance of the distribution which shows a considerable increase of the computational cost when the variance is increased. However, the number of instances that can be solved optimally within the time limit of 1000 seconds only slightly decreases to 376 for the largest variance setting we considered.

Impact of the number of scenarios. The number of scenarios that are to be considered is crucial for the performance of the branch-and-bound algorithms, because for a given node in the search tree we must compute earliest job start times for each scenario. In Figure 6.9 we show for different numbers of scenarios the average and maximum percental deviation of the expected makespan from the deterministic problem with $p = E[\boldsymbol{p}]$. For 200 scenarios – which we have chosen as default – the expected makespan varies only little when compared to larger sampling sizes. On the other hand, the computation time drastically increases with the number of scenarios. The average computation times depending on the number of scenarios is displayed in Figure 6.10.

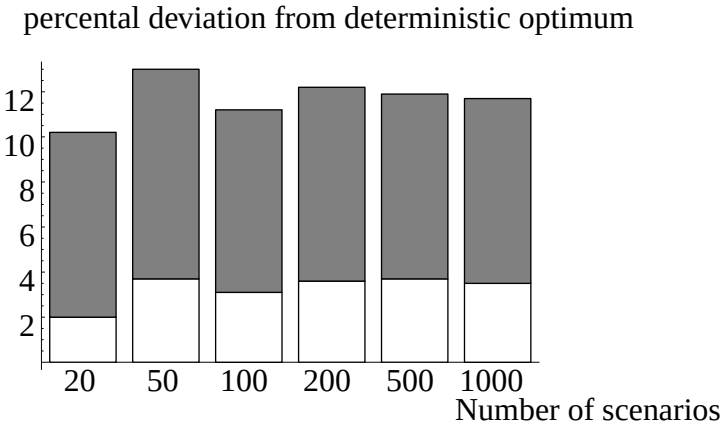

Figure 6.9: Average and maximum percental deviation of the expected makespan from the deterministic optimum (obtained with LIN-FS). The figures are based upon 344 instances that were solved optimally by all variations of the number of scenarios ($|P| \in \{20, 50, 100, 200, 500, 1000\}$).

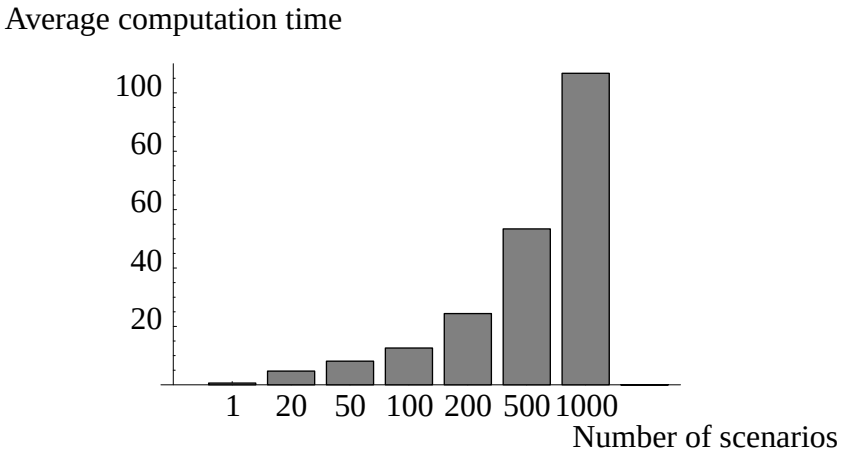

Figure 6.10: Average computation times (in seconds) depending on the number of scenarios (obtained with LIN-FS). The figures are based upon 344 instances that were solved optimally by all variations of the number of scenarios ($|P| \in \{1, 20, 50, 100, 200, 500, 1000\}$). For $|P| = 1$ the instance has deterministic processing times, we chose $E[\boldsymbol{p}]$.

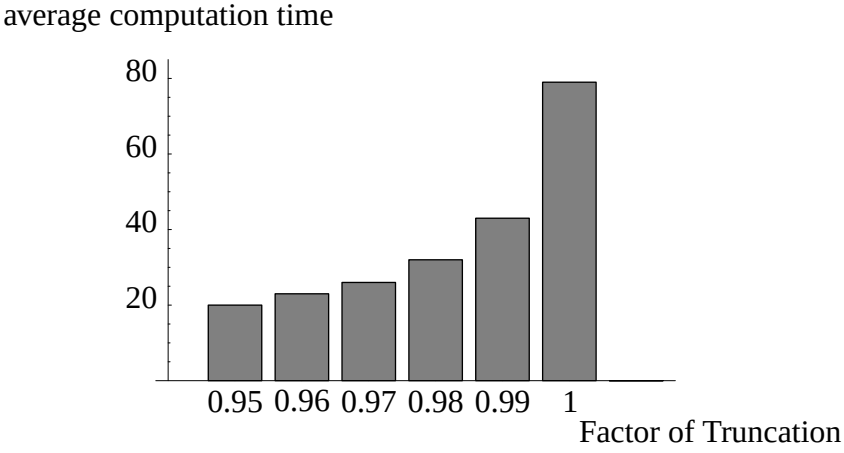

Figure 6.11: The computation time (in seconds) for different values of truncations, averaged over 390 instances that can be 'optimized' by all variations of truncation ($\alpha \in \{1.00, 0.99, 0.98, 0.97, 0.96, 0.95\}$).

Sensitivity of the cost function and truncation. We have performed several experiments where the branch-and-bound procedures are truncated. That is, for given $\alpha \in [0, 1]$ we remove nodes from the search if the lower bound computed for that node is larger than or equal to $\alpha \cdot ub$, where ub denotes the current global upper bound. We display the results for such truncated variations of the branch-and-bound algorithms in Figure 6.11. The data refers to the enumeration of linear preselective policies (LIN-FS). For different values of α (1.00, 0.99, 0.98, 0.97, 0.96, 0.95) we display the computation time averaged over the instances that can be 'optimized' by all variations of α. If we accept an optimality gap of 5% computation times can be reduced to one forth. In fact, for the 390 instances solved for both $\alpha = 1$ and $\alpha = 0.95$, the optimality gap was below 1% on average (3% maximum). Interestingly, there is a notedly large reduction of computation time between $\alpha = 1$ and $\alpha = 0.99$; it is almost halved when compared to the exact procedure. The reason is related to the fact that the objective function *expected makespan* is sensitive to minor (local) modifications of the considered scheduling policy. There are less policies with the same expected makespan and thus it is likely that more nodes in the search tree have to be evaluated. The impact of the sensitivity in terms of the number of nodes in the search tree is demonstrated by the following example. In the example we compare the cost function *expected makespan* to the deterministic counterpart.

Example 6.5.1. *Consider Example 2.2.1 and assume that all job processing times are independently distributed as follows:* $Pr(p_j = 7) = Pr(p_j = 13) = \frac{1}{2}$.

The processing time distributions of the jobs lead to 32 possible scenarios. Suppose that the selection $s = (5, 4, 4)$ is determined by some constructive heuris-

tic and serves as an initial upper bound. The bound is 20 for the deterministic problem with expected processing times $E[\boldsymbol{p}]$ and 22.8125 in the stochastic case. On the other hand, the critical path lower bound is 20 and 22.25, respectively. Consequently, branching is not required in the deterministic case; only the root node of the search tree is explored. In the stochastic case, however, eight nodes must be evaluated in order to prove optimality of s.

Sorting the forbidden sets. We finally report on the impact of ordering minimal forbidden sets within preprocessing (as described in Section 6.4.4). This preprocessing step turns out to be an important feature of the forbidden set branching scheme: For each of the forbidden set based branch-and-bound algorithms we can solve by far more instances to optimality within shorter computation times. For LIN-FS without this preprocessing step we solved 295 (out of 480) instances to optimality. This equals a loss of roughly 25% of optimized instances when compared to the experiment where sorting of forbidden sets was enabled (there, 390 instances were solved optimally). Moreover, the average computation time required to solve these instances increases by a factor of roughly 4. The differences for the other algorithms (PRS-FS and ES-FS) are of the same order of magnitude.

6.5.5 Application to other Instances

The experiments performed so far were restricted to instances of small size, i. e., the number of jobs in each instance was small. In this section we report on results that were obtained by applying each of the branch-and-bound algorithms to test sets of instances with 30 and 60 jobs, respectively. The test sets have already been described in Section 4.4.1; in total, there are 480 instances with 30 jobs and 480 instances with 60 jobs. In addition to the instances of the PSPLIB, we applied the branch-and-bound algorithms to an instance taken from (Golenko-Ginzburg and Gonik 1997).

For each of the experiments presented next we restricted the computation time to a maximum of 100 seconds per instance. The results for the test sets of the PSPLIB are displayed in Table 6.5. In the first column we state the used algorithm and in the second column we give the size of the considered instances (in terms of the number of jobs per instance). The third column shows the number of instances where the size of the initial data (the instance, the minimal forbidden sets and some additional data that is created within preprocessing) exceeded the limit of 50 MB. The figures in Columns 4 and 5 display the number of instances where the branch-and-bound was aborted due to the memory and time limit, respectively. The sixth column finally gives the number of instances that were solved to optimality. For instances with 30 jobs, although the number of minimal forbidden sets

Algorithm	#jobs	pre-process	memory limit	time limit	optimized
PRS-FS	30	0	0	338	142
LIN-FS	30	0	0	301	179
ES-FS	30	0	50	307	123
LIN-PT	30	0	0	379	101
JBP-PT	30	0	0	478	2
PRS-FS	60	71	127	280	2
LIN-FS	60	71	82	316	11
ES-FS	60	71	230	177	2
LIN-PT	60	71	2	394	7
JBP-PT	60	0	0	480	0

Table 6.5: Results of the algorithms applied to 480 instances with 30 and 60 jobs, respectively. The figures show the number of instances that had to be aborted due to the memory limit within preprocessing (Column 3), memory and time limit within the branch-and-bound (Columns 4 and 5), and the number of optimally solved instances (Column 6). We restricted the computation time to 100 seconds and the memory limit to 50 MB per instance.

is considerably larger when compared to the instances with 20 jobs (326 on average, 4411 maximum), for each of the 480 instances a feasible solution was found. LIN-FS solved 179 out of 480 instances to optimality, which are considerably more instances when compared to the other branch-and-bound algorithms. Moreover, on average over all 480 instances, LIN-FS produced the best feasible solutions, which are even slightly better than the solutions obtained from the preselective algorithm (recall that, contrarily, for the optimum values we have $\rho^{\mathrm{PRS}} \leqslant \rho^{\mathrm{LIN}}$). For the test set with 60 jobs per instance, we see that almost none of the instances was solved to optimality. Even LIN-FS can verify optimality for only 11 instances. The reason is that due to the very many minimal forbidden sets (often more than 20,000) all algorithms except for JBP-PT can evaluate only few nodes of the search tree. In particular, for 110 instances with more than 20,000 minimal forbidden sets each, the average number of nodes that are evaluated within a second is 16 for LIN-FS. For instances with less than $20,000$ minimal forbidden sets (299), 38 nodes are evaluated per second (these figures are based on 200 scenarios of job processing times). JBP-PT evaluates roughly 28 nodes per second, independently of the number of minimal forbidden sets. Consequently, for instances with many minimal forbidden sets the improvement of the expected makespan is negligible when compared to the initial upper bound. Contrarily, JBP-PT improves the initial upper bound by roughly 2.5% on average (13% maximum).

Finally, in addition to the above instances, we considered a project with 36 jobs

taken from (Golenko-Ginzburg and Gonik 1997). In contrast to the instances of the PSPLIB, this instance already includes information of random job processing times, that is, for each job j, a minimum and maximum processing time p_j^{min} and p_j^{max} is given. We then assume that each processing time is uniformly distributed. The uniform distribution was also considered in (Golenko-Ginzburg and Gonik 1997). The instance contains 3730 minimal forbidden sets. Moreover, assuming fixed job processing times $p_j = (p_j^{min} + p_j^{max})/2$, a deterministic upper bound of 419 was computed by the algorithm JBP-PT.

Golenko-Ginzburg and Gonik (1997) compute a feasible solution for that instance with expected makespan 448 (we rounded all reported values appropriately). Recall from the introduction of the chapter that, at each job completion time t, they solve an NP-hard knapsack type problem in order to define sets of jobs to be started. They also suggest to heuristically compute the sets which resulted in a solution of (rounded) 461. They do not report on computation times of the heuristics. In fact, already the starting solutions of our algorithms (see Section 6.4.1) are of comparable quality; in particular, the initial job-based priority policy has an expected makespan of (rounded) 445; computation time is negligible. Moreover, JBP-PT constructs a solution with an expected makespan of (rounded) 434 in less than 40 seconds. However, the other algorithms were not able to improve their initial solution within a time limit of 100 seconds.

To conclude this section, although JBP-PT behaved poorly for verifying optimality, for the considered instances, the algorithm works quite reasonable if the goal is to compute feasible solutions of good quality.

CONCLUDING REMARKS

The purpose of this thesis was to develop theory and algorithms which lead to a better understanding of stochastic resource-constrained project scheduling problems. Starting from the observation that the widely used class of priority policies is not a good choice in terms of a 'robust' execution of projects, we focussed on the structurally appealing class of preselective policies which does not show the so-called Graham anomalies. We first studied the model of AND/OR precedence constraints which turned out to be the combinatorial core of a preselective policy and we established various results that are useful within their theoretical and computational treatment. We then focussed on the representation of resource constraints by minimal forbidden sets which are necessary to define and handle preselective policies. We developed an effective algorithm to construct all minimal forbidden sets of a given instance which in addition suggests a considerably compact representation.

The above summarized theory suggested to additionally study the subclasses of linear preselective policies and job-based priority policies which are computationally more tractable in comparison to preselective policies. Based on these classes of policies we developed branch-and-bound algorithms and established results on the trade-off between computational efficiency on the one hand and solution quality on the other hand. Moreover, we explored the practical limits of forbidden set based policies. The implemented branch-and-bound algorithms exposed that good feasible solutions for instances with less than 1,000 minimal forbidden sets can easily be computed; for instances with less than 100 minimal forbidden sets the algorithms even computed optimal solutions (in the considered class) within short computation times. Notice that with additional truncation (or rounding of expected cost values) the above mentioned limits on the number of forbidden sets can be further increased. Moreover, by appropriate preprocessing, the number of minimal forbidden sets in the instance can be reduced considerably (e. g., by inserting additional precedence constraints). However, a very large number of minimal forbidden sets makes the use of forbidden set based solution procedures computationally inefficient. Then, the remaining alternative (among the classes of policies considered in this thesis) is to use job-based priority policies since they do not require the representation of resource-constraints by minimal forbidden sets. Although the minimum expected cost that can be achieved in the class of job-based priority policies may be considerably larger when compared to

(linear) preselective policies, we have empirically demonstrated that for the considered instances the deviation was only marginal: the optimum makespan among the class of job-based priority policies was only slightly larger (less than 1% on average over 149 instances) when compared to the optimum makespan within the class of preselective policies. Hence, the class of job-based priority policies (but also the class of linear preselective policies) seems to be an attractive starting point to develop heuristic algorithms for computing 'robust' solutions of good quality.

The contributions of this thesis not only seem to lead to a better understanding of resource-constrained project scheduling problems, they also raise a number of interesting questions and future research directions.

First, given a set of AND/OR precedence constraints with arbitrary time lags between the start times of jobs, does there exist a polynomial time algorithm to decide whether there exists a feasible schedule? The problem has not only received attention within the context of scheduling problems, it also occurs within other fields such as, e. g., game theory and online optimization and is hence of strong interest within a broader context. Since the problem is in NP $\cap$ co-NP, and since there exists a simple pseudo-polynomial time algorithm to solve the problem, we believe that there exists a polynomial time algorithm. This opinion is shared by Zwick and Paterson (1996) who conjecture in their concluding remarks that deciding a mean payoff game is in P. We think that Lemma 3.2.3 may be a useful starting point to derive new insights towards a polynomial time algorithm. It follows from the lemma that it suffices to decide in polynomial time whether there exists a generalized cycle in a digraph of AND/OR precedence constraints which contains no (ordinary) cycle of non-positive length.

The second topic we want to address is related to resource-constrained project scheduling instances with very scarce resources. Many previously studied algorithms for deterministic resource-constrained project scheduling problems require very large computation times for problems where resources are scarce. We observed that there often exist surprisingly few minimal forbidden sets for instances with very scarce resources. Moreover, the forbidden set based branch-and-bound algorithms showed quite reasonable performance when applied to instances with this property. Hence, it may be fruitful to develop algorithms which take advantage of the structural benefits of minimal forbidden sets to solve instances with very scarce resources.

Another interesting field of research is concerned with a more detailed analysis of a given policy. Suppose that it has already been decided that a project is executed according to the policy Π. For risk management purposes it is then important to collect more detailed information on the expected start times of jobs as well as on the evolution of project costs. Hence, methods are required to (approximately) compute the entire *project cost distribution*. It is a strong benefit

of ES-policies that they can be represented by a stochastic project network. For such networks, the determination of the project cost distribution is a well studied problem as we have noted in Chapter 1.2. If Π is a preselective policy, however, then the results on stochastic project networks are not directly applicable. The problem is to generalize methods developed for traditional precedence constraints (ES-policies, stochastic project networks) to AND/OR precedence constraints (preselective policies). In fact, this seems sometimes possible as the following example demonstrates. To compute stochastic bounds on the makespan distribution of a linear preselective policy, one can basically apply an algorithm for traditional precedence constraints proposed by Kleindorfer (1971). The core problem is to compute the maximum of possibly dependent random variables. Based on a topological ordering of the jobs, Kleindorfer (1971) computes stochastic (lower and upper) bounds on the maximum $\max_{i \in Pred_j}(C_i(\boldsymbol{p}))$ of job completion times which yield bounds on the completion time distribution of each job j. Using similar stochastic bounds on the *minimum* of (possibly dependent) random variables and the fact that a linear preselective policy can be expressed as a topological ordering of the jobs, we obtain bounds on the makespan distribution of a linear preselective policy by essentially the same approach.

LIST OF ALGORITHMS

BIBLIOGRAPHY

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.

Adlakha, V. G. and V. G. Kulkarni (1989). A classified bibliography of research on stochastic PERT networks: 1966-1987. *INFOR 27*, 272–296.

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin (1993). *Network Flows*. Prentice Hall.

Alvarez-Valdés Olaguíbel, R. and J. M. Tamarit Goerlich (1993). The project scheduling polyhedron: Dimension, facets, and lifting theorems. *European Journal of Operational Research 67*, 204–220.

Arnold, B. C. (1988). Bounds on the expected maximum. *Communications in Statistics, Theory and Methods 17*, 2135–2150.

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., A. d'Atri, and D. Saccà (1983). Graph algorithms for functional dependency manipulation. *Journal of the ACM 30*, 752–766.

Ausiello, G., A. d'Atri, and D. Saccà (1986). Minimal representations of directed hypergraphs. *SIAM Journal on Computing 15*, 418–431.

Balas, E. (1971). Project scheduling with resource constraints. In E. M. L. Beale (Ed.), *Applications of Mathematical Programming*, pp. 187–200. The English University Press, London.

Balas, E. and E. Zemel (1978). Facets of the knapsack polytope from minimal covers. *SIAM Journal on Applied Mathematics 34*, 119–148.

Bartusch, M. (1984). *Optimierung von Netzplänen mit Anordnungsbeziehungen bei knappen Betriebsmitteln*. Ph. D. thesis, Rheinisch-Westfälische Technische Hochschule Aachen.

Bartusch, M., R. H. Möhring, and F. J. Radermacher (1988). Scheduling project networks with resource constraints and time windows. *Annals of Operations Research 16*, 201–240.

Bast, H. (1998). Dynamic scheduling with incomplete information. In *Proc. 10th Annual Symposium on Parallel Algorithms and Architectures SPAA, Puerto Vallerta, Mexico*, pp. 182 – 191. ACM.

Birge, J. R. and M. A. H. Dempster (1996). Stochastic programming approaches to stochastic scheduling. *Journal of Global Optimization 9*, 383–409.

Bouleimen, K. and H. Lecocq (2000). A new efficient simulated annealing algorithm for the resource constrained project scheduling problem and its multiple modes version. Preprint, Service de Robotique et Atomatisation, Université de Liège, Liège, Belgium.

Brandstädt, A., V. B. Le, and J. P. Spinrad (1999). *Graph Classes: A Survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics.

Bratley, P., B. L. Fox, and L. E. Schrage (1987). *A Guide to Simulation* (2nd ed.). Springer-Verlag.

Brucker, P. (1998). *Scheduling Algorithms*. Springer-Verlag.

Brucker, P., A. Drexl, R. H. Möhring, K. Neumann, and E. Pesch (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research 112*, 3–41.

Brucker, P. and S. Knust (2000). A linear programming and constraint propagation-based lower bound for the RCPSP. *European Journal of Operational Research 127*, 355–362.

Brucker, P., S. Knust, A. Schoo, and O. Thiele (1998). A branch & bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research 107*, 272–288.

Bruno, J. L., P. J. Downey, and G. N. Frederickson (1981). Sequencing tasks with exponential service times to minimize the expected flowtime or makespan. *Journal of the Association for Computing Machinery 28*, 100–113.

Carlier, J. (1982). The one–machine sequencing problem. *European Journal of Operational Research 11*, 42–47.

Cavalcante, C. C. B., C. C. de Souza, M. W. P. Savelsbergh, Y. Wang, and L. A. Wolsey (1998). Scheduling projects with labour constraints. Technical Report 9859, CORE discussion paper.

Chauvet, F. and J.-M. Proth (1999). The PERT-problem with alternatives: Modelisation and optimisation. Technical report, Institut National de Recherche en Informatique et en Automatique (INRIA), France.

Christofides, N., R. Alvarez-Valdes, and J. M. Tamarit (1987). Project scheduling with resource constraints: A branch-and-bound approach. *European Journal of Operational Research 29*, 262–273.

Chvátal, V. and P. L. Hammer (1977). Aggregation of inequalities in integer programming. *Annals of Discrete Mathematics 1*, 145–162.

Cook, W., W. Cunnigham, W. Pulleyblank, and A. Schrijver (1998). *Combinatorial Optimization*. Wiley.

Demeulemeester, E. and W. Herroelen (1992). A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science 38*, 1803–1818.

Demeulemeester, E. and W. Herroelen (1997). New benchmark results for the resource-constrained project scheduling problem. *Management Science 43*, 1485–1492.

Devroye, L. P. (1979). Inequalities for the completion times of stochastic PERT networks. *Mathematics of Operations Research 4*, 441–447.

Dinic, E. A. (1990). The fastest algorithm for the PERT problem with AND- and OR-nodes (the new-product-new technology problem). In R. Kannan and W. R. Pulleyblank (Eds.), *Integer Programming and Combinatorial Optimization*, Proceedings of a conference held at the University of Waterloo, May 28-30, 1990, by the Mathematical Programming Society, pp. 185–187.

Dodin, B. (1985). Bounding the project completion time distribution in PERT networks. *Operations Research 33*, 862–881.

Dorndorf, U., E. Pesch, and T. Phan Huy (2000a). A branch-and-bound algorithm for the resource-constrained project scheduling problem. *Mathematical Methods of Operations Research 52*, 413–439.

Dorndorf, U., E. Pesch, and T. Phan Huy (2000b). A time oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints. *Management Science 46*, 1365–1384.

Dowling, W. F. and J. H. Gallier (1984). Linear-time algorithms for testing satisfiability of propositional Horn formulae. *Journal on Logic Programming 1*, 267–284.

Duchet, P. (1995). Hypergraphs. In R. Graham, M. Grötschel, and L. Lovász (Eds.), *Handbook of Combinatorics*, Chapter 7, pp. 381–432. Amsterdam: Elsevier Science.

Ehrenfeucht, A. and J. Mycielski (1979). Positional strategies for Mean Payoff Games. *International Journal of Game Theory 8*, 109–113.

Escudero, L. F., P. V. Kamesam, A. J. King, and R. J.-B. Wets (1993). Production planning via scenario modelling. *Annals of Operations Research 43*, 311–335.

Feige, U. and J. Kilian (1998). Zero-knowledge and the chromatic number. *Journal of Computer and System Sciences 57*, 187–199.

Fernandez, A. A. and R. L. Armacost (1996). The role of the non-anticipativity constraint in commercial software for stochastic project scheduling. *Computers and industrial engineering 31*, 233–236.

Fernandez, A. A., R. L. Armacost, and J. Pet-Edwards (1998a). A model for the resource constrained project scheduling problem with stochastic task durations. In *Proceedings of the 7th Annual Industrial Engineering Research Conference*, Banff, Alberta, Canada.

Fernandez, A. A., R. L. Armacost, and J. Pet-Edwards (1998b). Understanding simulation solutions to resource constrained project scheduling problems with stochastic task durations. *Engineering Management Journal 10*, 5–13.

Fest, A., R. H. Möhring, F. Stork, and M. Uetz (1998). Resource-constrained project scheduling with time windows: A branching scheme based on dynamic release dates. Technical Report 596/1998, Technische Universität Berlin, Department of Mathematics, Germany. Revised 1999.

Fulkerson, D. R. (1961). A network flow computation for project cost curves. *Management Science 7*, 167 – 178.

Fulkerson, D. R. (1962). Expected critical path length in PERT networks. *Operations Research 10*, 808–817.

Gallo, G., C. Gentile, D. Pretolani, and G. Rago (1998). Max Horn SAT and the minimum cut problem in 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.

Garey, M. J. and D. S. Johnson (1979). *Computers and Intractibility: A Guide to the Theory of $\mathcal{NP}$-Completeness*. New York: Freemann.

Gemmil, D. D. (2000). Personal communication.

Gillies, D. W. (1993). *Algorithms to schedule tasks with AND/OR precedence constraints*. Ph. D. thesis, University of Illinois at Urbana-Champaign, Department of Computer Science, Urbana, Illinois, USA.

Gillies, D. W. and J. W.-S. Liu (1995). Scheduling tasks with AND/OR precedence constraints. *SIAM Journal on Computing 24*, 797–810.

Goldwasser, M. H. and R. Motwani (1999). Complexity measures for assembly sequences. *International Journal of Computational Geometry and Applications 9*, 371–418.

Golenko-Ginzburg, D. and A. Gonik (1997). Stochastic network project scheduling with non-consumable limited resources. *International Journal of Production Economics 48*, 29–37.

Golumbic, M. C. (1980). *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York.

Graham, R. L. (1966). Bounds on multiprocessing timing anomalies. *Bell System Technical Journal 45*, 1563–1581.

Hagstrom, J. N. (1988). Computational complexity of PERT problems. *Networks 18*, 139–147.

Hartmann, S. (1999). Self-adapting genetic algorithms with an application to project scheduling. Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel 506, Christian-Albrechts-Universität zu Kiel, Germany.

Hartmann, S. and R. Kolisch (1998). Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In J. Węglarz (Ed.), *Handbook on Recent Advances in Project Scheduling*. Kluwer, Amsterdam.

Heilmann, R. and C. Schwindt (1997). Lower bounds for RCPSP/max. Technical Report 511, WIOR, University of Karlsruhe, Germany.

Henderson, P. B. and Y. Zalcstein (1977). A graph-theoretic characterization of the PV_{chunk} class of synchronizing primitives. *SIAM Journal on Computing 6*, 88–108.

Henrion, M., R. Fung, T. Cheung, M. Steele, and B. Basevich (1996). Integrated risk analysis for schedule and cost. Technical Report NAS10-12116, NASA/Kennedy Space Center, performing organization: Lumina Decision Systems Inc.

Igelmund, G. and F. J. Radermacher (1983a). Algorithmic approaches to preselective strategies for stochastic scheduling problems. *Networks 13*, 29–48.

Igelmund, G. and F. J. Radermacher (1983b). Preselective strategies for the optimization of stochastic project networks under resource constraints. *Networks 13*, 1–28.

Jurdziński, M. (1998). Deciding the winner in parity games is in UP $\cap$ co-UP. *Information Processing Letters 68*, 119–124.

Kaerkes, R., R. H. Möhring, W. Oberschelp, F. J. Radermacher, and M. M. Richter (1981). Mathematsche Untersuchungen zur stochastischen Kapazitätsoptimierung. Report on a research project at the RWTH Aachen, Germany.

Kämpke, T. (1985). *Optimalitätsaussagen für spezielle stochastische Schedulingprobleme*. Ph. D. thesis, RWTH Aachen, Germany.

Kelley, J. E. (1961). Critical path planning and scheduling: Mathematical basis. *Operations Research 9*, 296 – 320.

Kelley, J. E. and M. R. Walker (1959). Critical path planning and scheduling: An introduction. Mauchly Associates, Inc., Ambler, Pa.

Klein, R. and A. Scholl (1999). Computing lower bounds by destructive improvement: An application to resource–constrained project scheduling. *European Journal of Operational Research 112*, 322–346.

Kleindorfer, G. B. (1971). Bounding distributions for a stochastic acyclic network. *Operations Research 19*, 1586–1601.

Kleywegt, A. J. and A. Shapiro (1999). The sample average approximation method for stochastic discrete optimization. Technical report, School of Industrial and Systems Engineering, Georgia Institute of Technology.

Knuth, D. E. (1977). A generalization of Dijkstra's algorithm. *Information Processing Letters 6*, 1–5.

Kolisch, R. (1996). Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research 90*, 320–333.

Kolisch, R. and A. Sprecher (1996). PSPLIB - a project scheduling problem library. *European Journal of Operational Research 96*, 205–216.

Korte, B., L. Lovász, and R. Schrader (1991). *Greedoids*. Springer-Verlag.

Korte, B. and J. Vygen (2000). *Combinatorial Optimization: Theory and Complexity*. Springer-Verlag.

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*, pp. 445–522. North-Holland, Amsterdam.

Levner, E., S. C. Sung, and M. Vlach (1999). Multiple-choice project scheduling. Technical report, Japan Advanced Institute of Science and Technology, Hokuriku.

Levner, E., S. C. Sung, and M. Vlach (2000). On Project Scheduling with alternatives. In *Proceedings of the 7th International Workshop on Project Management and Scheduling*, Osnabrück, Germany, pp. 197–198.

Ludwig, W. (1995). A subexponential randomized algorithm for the simple stochastic game problem. *Information and Computation 117*, 151–155.

Mahadev, N. V. R. and U. N. Peled (1995). *Threshold Graphs and Related Topics*, Volume 56 of *Annals of Discrete Mathematics*. North-Holland.

Malcom, D. G., J. H. Roseboom, C. E. Clark, and W. Fazar (1959). Application of a technique for research and development program evaluation. *Operations Research 7*, 646–669.

Martin, J. J. (1965). Distribution of the time through a directed acyclic network. *Operations Research 13*, 46–66.

McMillan, K. L. and D. L. Dill (1992). Algorithms for interface timing veryfication. In *IEEE International Conference on Computer Design*, pp. 48–51.

Mingozzi, A., V. Maniezzo, S. Ricciardelli, and L. Bianco (1998). An exact algorithm for the multiple resource-constraint project scheduling problem based on a new mathematical formulation. *Management Science 44*, 714–729.

Möhring, R. H. (1985). Algorithmic aspects of comparability graphs and interval graphs. In I. Rival (Ed.), *Graphs and Order*, pp. 41–101. D. Reidel Publishing Company, Dordrecht.

Möhring, R. H. (2000a). Scheduling under uncertainty: Bounding the makespan distribution. Technical Report 610/1998, Technische Universität Berlin, Department of Mathematics, Germany. To appear in Springer Lecture Notes in Computer Science.

Möhring, R. H. (2000b). Scheduling under uncertainty: Optimizing against a randomizing adversary. In *Proceedings of the 3nd International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, Lecture Notes in Computer Science, Saarbrücken, Germany. Springer-Verlag.

Möhring, R. H. and R. Müller (1998). A combinatorial approach to bound the distribution function of the makespan in stochastic project networks. Technical Report 610/1998, Technische Universität Berlin, Department of Mathematics, Germany.

Möhring, R. H. and F. J. Radermacher (1985). Introduction to stochastic scheduling problems. In K. Neumann and D. Pallaschke (Eds.), *Contributions to Operations Research, Proceedings of the Oberwolfach Conference*

on Operations Research, 1984, pp. 72–130. Springer-Verlag, Lecture Notes in Economics and Mathematical Systems, vol. 240.

Möhring, R. H., F. J. Radermacher, and G. Weiss (1984). Stochastic scheduling problems I: General strategies. *ZOR – Zeitschrift für Operations Research 28*, 193–260.

Möhring, R. H., F. J. Radermacher, and G. Weiss (1985). Stochastic scheduling problems II: Set strategies. *ZOR – Zeitschrift für Operations Research 29*, 65–104.

Möhring, R. H., A. S. Schulz, F. Stork, and M. Uetz (2000). Solving project scheduling problems by minimum cut computations. Technical Report 680, Technische Universität Berlin, Department of Mathematics. Submitted.

Möhring, R. H., A. S. Schulz, and M. Uetz (1999). Approximation in stochastic scheduling: The power of LP-based priority policies. *Journal of the Association for Computing Machinery 46*, 924–942.

Möhring, R. H., M. Skutella, and F. Stork (2000a). Forcing relations for AND/OR precedence constraints. In *Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00)*, pp. 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. A preliminary version of this paper (*Forcing Relations for AND/OR Precedence Constraints*) appeared in (Möhring, Skutella, and Stork 2000a).

Möhring, R. H. and F. Stork (2000). Linear preselective policies for stochastic project scheduling. *Mathematical Methods of Operations Research 52*, 501–515.

Mulvey, J. M., R. J. Vanderbei, and S. A. Zenios (1995). Robust optimization of large-scale systems. *Operations Research 43*, 264–281.

Nilsson, N. J. (1980). *Principals of Artificial Intelligence*. Palo Alto, USA: Tioga Publishing Company.

Ordman, E. T. (1987). Dining philosophers and graph covering problems. *The Journal of Combinatorial Mathematics and Combinatorial Computing 1*, 181–190.

Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley Publishing Company.

Papadimitriou, C. H. and K. Steiglitz (1982). *Combinatorial Optimization; Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs.

Patterson, J. H. (1984). A comparison of exact approaches for solving the multiple constrained resource project scheduling problem. *Management Science 30*, 854–867.

Patterson, J. H., R. Słowiński, F. B. Talbot, and J. Węglarz (1989). An algorithm for a general class of precedence and resource constrained scheduling problems. In R. Słowiński and J. Węglarz (Eds.), *Advances in Project Scheduling*, pp. 3–28. Elsevier.

Phillips, S. and M. I. Dessouky (1977). Solving the project time/cost tradeoff problem using the minimal cut concept. *Management Science 24*, 393 – 400.

Pinedo, M. (1995). *Scheduling: Theory, algorithms and systems*. International Series in Industrial and Systems. Prentice Hall.

Provan, J. S. and M. O. Ball (1983). The complexity of counting cuts and of the probability that a graph is connected. *SIAM J. Comput. 12*, 777–788.

PSPLIB (2000). `ftp://ftp.bwl.uni-kiel.de/pub/operations-research/psplib/HTML/data.html`.

Radermacher, F. J. (1981a). Cost-dependent essential systems of ES-strategies for stochastic scheduling problems. *Methods of Operations Research 42*, 17–31.

Radermacher, F. J. (1981b). Optimale Strategien für stochastische Scheduling Probleme. Habilitationsschrift, RWTH Aachen. In: Schriften zur Informatik und angewandten Mathematik 98, RWTH Aachen, 1984.

Radermacher, F. J. (1985). Scheduling of project networks. *Annals of Operations Research 4*, 227–252.

Radermacher, F. J. (1986). Analytical vs. combinatorial characterizations of well-behaved strategies in stochastic scheduling. *Methods of Operations Research 53*, 467–475.

Raz, R. and S. Safra (1997). A sub-constant error-probability low-degree test, and sub-constant error-probability pcp characterization of np. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing*, pp. 475–484.

Righter, R. (1994). Scheduling. In *Stochastic orders and their applications*, Chapter 13, pp. 381–432. Academic Press.

Rockafellar, R. T. and R. J.-B. Wets (1991). Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research 16*, 119–147.

Ross, S. (1983). *Inroduction to stochastic dynamic programming*. Academic Press.

Schäffter, M. (1997). Scheduling with respect to forbidden sets. *Discrete Applied Mathematics 72*, 141–154.

Schrijver, A. (1986). *Theory of Linear and Integer Programming*. John Wiley & Sons.

Schwiegelshohn, U. and L. Thiele (1999). Dynamic min-max problems. *Discrete Event Dynamic Systems 9*, 111–134.

Schwindt, C. (1998). A branch-and-bound algorithm for the resource-constrained project duration problem subject to temporal constraints. Technical Report 544, WIOR, University of Karlsruhe, Germany.

Skutella, M. (1998). *Approximation and Randomization in Scheduling*. Ph. D. thesis, Technische Universität Berlin, Germany.

Skutella, M. and M. Uetz (2001). Scheduling precedence-constrained jobs with stochastic processing times on parallel machines. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*. to appear.

Sotskov, Y. N., V. S. Tanaev, and F. Werner (1998). On the stability radius of an optimal schedule: a survey and recent developments. In *Industrial Applications of Combinatorial Optimization*, Volume 16, pp. 72–108. Boston, MA, USA: Kluwer Academic Publishers.

Sprecher, A. (2000). Scheduling resource-constrained projects competitively at modest memory requirements. *Management Science 46*, 710–723.

Stinson, J. P., E. W. Davis, and B. H. Khumawala (1978). Multiple resource constrained scheduling using branch and bound. *AIEE Transactions 10*, 252–259.

Stork, F. (2000). Branch-and-bound algorithms for stochastic resource-constrained project scheduling. Technical Report 702/2000, Technische Universität Berlin, Department of Mathematics, 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: Dimension Theory*. John Hopkins University Press, Baltimore, ML.

Tsai, Y.-W. and D. D. Gemmill (1998). Using tabu search to schedule activities of stochastic resource-constrained projects. *European Journal of Operational Research 111*, 129–141.

Ullman, J. D. (1982). *Principles of database systems*. Computer Science Press, Inc.

Valls, V., M. Laguna, P. Lino, A. Pérez, and M. S. Quintanilla (1998). Project scheduling with stochastic activity interruptions. In J. Węglarz (Ed.), *Handbook on Recent Advances in Project Scheduling*, pp. 333–353. Kluwer, Amsterdam.

van Slyke, R. M. (1963). Monte Carlo methods and the PERT problem. *Operations Research 11*, 839–860.

Vöge, J. and M. Jurdziński (2000). A discrete strategy improvement algorithm for solving parity games. In E. A. Emerson and A. P. Sistla (Eds.), *Computer Aided Verification, 12th International Conference, CAV 2000, Proceedings*, Volume 1855 of *Lecture Notes in Computer Science*, Chicago, Illinois, USA, pp. 202–215. Springer-Verlag.

Węglarz, J. (Ed.) (1999). *Project Scheduling: Recent Models, Algorithms, and Applications*. Kluwer.

Wets, R. J.-B. (1989). The aggregation principle in scenario analysis and stochastic optimization. In S. W. Wallace (Ed.), *Algorithms and Model Formulations in Mathematical Programming*, Volume F51 of *Nato ASI Series*, pp. 91–113. Springer-Verlag.

Wiest, J. D. (1963). Some properties of schedules for large projects with limited resources. *Operations Research 12*, 395–418.

Yannakakis, M. (1982). The complexity of the partial order dimension problem. *SIAM Journal on Algebraic and Discrete Methods 3*, 351–358.

Zwick, U. and M. Paterson (1996). The complexity of Mean Payoff Games on graphs. *Theoretical Computer Science 158*, 343–359.

SYMBOL INDEX

CP	the critical path lower bound, page 103
C_j	the completion time of job j, page 7
C	a vector $(C_1, \ldots, C_n)$ of job completion times, page 8
E_0	the set of precedence constraints, page 7
E	a set of precedence constraints with $E_0 \subseteq E$
$\mathcal{F}$	the set of minimal forbidden sets, page 52
f	the number of forbidden sets, page 79
$F \in \mathcal{F}$	a minimal forbidden set, page 52
G_0	the partial order induced by the precedence constraints E_0, page 7
$i, j, h \in V$	jobs
κ	the cost function, page 7
K	the set of different resources, page 7
$k \in K$	a single resource, page 7
p_j	the processing time of job j, page 7
p	a scenario, a vector of job processing times, page 11
$\boldsymbol{p}_j$	the random variable of the processing time of job j, page 10
$\boldsymbol{p}$	the random vector of job processing times, page 11
P	a set of scenarios p of the random vector $\boldsymbol{p}$, page 103
Π	a policy, page 75
$Pred_j$	the set of predecessors of job j with respect to E_0, page 107
ρ^τ	the optimum expected cost value among the policy class τ , page 76
R_k	availability of resource k, page 7
r_{jk}	resource requirement of job j w.r.t. resource k, page 7
$\mathbb{R}_{\geqslant}$	the set of non-negative real numbers, page 7
$\mathbb{R}_{>}$	the set of positive real numbers, page 7
s	a selection $s = (s_1, \ldots, s_f)$, page 79
S_j	the start time of job j, page 7
S	a vector $(S_1, \ldots, S_n)$ of job start times, page 7
τ	a particular class of scheduling policies , page 76
V	the set of jobs, page 7
$w = (X, j)$	a waiting condition with predecessor set X and waiting job j, page 19
$\mathcal{W}$	the set of waiting conditions, page 17

INDEX

ZUSAMMENFASSUNG

In der ressourcenbeschränkten Projektplanung müssen Vorgänge (Jobs) unter Berücksichtigung von Reihenfolgebeziehungen und Kapazitätsrestriktionen zeitlich so eingeplant werden, dass eine gegebene Kostenfunktion minimiert wird. In der vorliegenden Arbeit wird zusätzlich davon ausgegangen, dass die Dauer der einzelnen Vorgänge nicht zu Beginn der Planung bekannt, sondern durch je eine Zufallsvariable gegeben ist. Auf diese Weise ist es möglich, unvorhersehbare Ereignisse wie zum Beispiel Wetterbedingungen, Krankheit von Mitarbeitern, Rechtsfragen oder Genehmigungsverfahren in die Planung eines Projekts mit einzubeziehen. Als Konsequenz hieraus kann das Risiko von Projekt-Verzögerungen und damit verbundenen Kostensteigerungen (von denen im Rahmen von umfangreichen Projekten häufig berichtet wird) reduziert werden. Dieses Modell wird in der Literatur häufig als *Stochastic Resource-Constrained Project Scheduling Problem* bezeichnet.

Ziel der vorliegenden Arbeit ist es, Algorithmen zu entwickeln, die die Berechnung guter Lösungen in akzeptabler Rechenzeit ermöglichen. Eine Lösung ist hier eine so genannte *Politik*, eine Planungsvorschrift, die zu jedem möglichen Entscheidungszeitpunkt t während der Umsetzung des Projekts eine Menge von Vorgängen definiert, deren Durchführung zum Zeitpunkt t begonnen werden soll. Basierend auf der Beobachtung, dass die häufig angewendeten *Prioritäts-Politiken* nicht für eine „robuste“ Planung geeignet sind, liegt der Schwerpunkt der Untersuchungen auf der strukturell sehr attraktiven Klasse der *präselektive Politiken*. Solche Politiken selektieren für jede *minimal verbotene Menge* F von Vorgängen einen Vorgang $j \in F$, dessen Durchführung erst dann begonnen wird, wenn mindestens ein anderer Vorgang $i \in F \setminus \{j\}$ beendet ist. Eine minimal verbotene Menge ist eine minimale Teilmenge von Vorgängen, zwischen denen keine Reihenfolgebeziehungen vorliegen, die jedoch aufgrund der Ressourcen-Beschränkungen nicht zur gleichen Zeit in Betrieb sein können.

Um ein bestmögliches Verständnis von präselektiven Politiken zu erzielen, betrachten wir im ersten Teil der vorliegenden Arbeit eine Verallgemeinerung klassischer Reihenfolgebeziehungen zwischen Vorgängen, die so genannten AND/OR Reihenfolgebeziehungen. Diese finden zum Beispiel im Rahmen von Montage- oder Demontage-Prozessen Anwendung; der Zusammenhang zu dem oben beschriebenen ressourcenbeschränkten Scheduling-Modell besteht in der Tatsache, dass präselektive Politiken durch eine Menge von AND/OR Reihenfolgebeziehun-

gen repräsentiert werden können. Es werden Resultate im Zusammenhang mit der Verallgemeinerung von Begriffen wie *transitive Hülle* und *transitive Reduktion* erzielt. Darüber hinaus werden für gegebene zeitliche Abstände zwischen den Vorgängen Algorithmen zur Berechnung frühster Startzeiten von Vorgängen entwickelt.

Da präselektive Politiken die explizite Angabe aller minimal verbotenen Mengen benötigen, diese jedoch üblicherweise nur implizit durch den Ressourcen-Bedarf der Vorgänge und die globale Ressourcen-Verfügbarkeit gegeben sind, wird ein Algorithmus entwickelt, der aus dieser impliziten Repräsentation alle minimal verbotenen Mengen errechnet. Es wird gezeigt, dass die Laufzeit dieses Algorithmus polynomial in der Kodierungslänge der Ein- und Ausgabe ist, falls die Ressourcen-Beschränkungen durch *einen* Ressource-Typ charakterisiert werden. Darüber hinaus wird ein Bezug zu so genannten *Threshold (Hyper-)Graphen* diskutiert.

Basierend auf den beschriebenen Ergebnissen werden Dominanzresultate für präselektive Politiken abgeleitet, sowie zwei Teilklassen der präselektiven Politiken definiert und untersucht, die vom algorithmischen Standpunkt gesehen bessere Eigenschaften als präselektive Politiken besitzen. Für diese Klassen von Politiken (und eine weitere, aus der Literatur bekannte Klasse) werden insgesamt fünf verschiedene Branch-and-Bound Verfahren entwickelt und implementiert sowie auf Basis von 1440 Instanzen getestet und miteinander verglichen. Die empirischen Ergebnisse zeigen, dass die entwickelten Resultate auf dem Gebiet der AND/OR Reihenfolgebeziehungen die Leistung der Branch-and-Bound Verfahren deutlich steigern und dass die vorgeschlagenen Teilklassen der präselektiven Politiken gute Ausgangspunkte für den algorithmischen Zugang zur heuristischen Lösung von Real-Life Instanzen darstellen.

CURRICULUM VITAE

14. Mai 1970	Geboren in München
1976 – 1982	Besuch der Grundschule am Fließtal in Berlin, Bezirk Reinickendorf
1982 – 1989	Besuch des Georg-Herwegh-Gymnasiums in Berlin, Bezirk Reinickendorf
1989	Abitur
1989 – 1996	Mitarbeiter der Softwarefirma C.I.T. GmbH, Berlin
1989 – 1996	Studium der Techno- und Wirtschaftsmathematik (Studienrichtung Technomathematik) an der Technischen Universität Berlin
1992	Vordiplom (Technomathematik)
1996	Diplom (Technomathematik)
1997 – 2001	Wissenschaftlicher Mitarbeiter an der Technischen Universität Berlin im Projekt “Projektplanung mit variablen Vorgangsdauern”, gefördert durch die Deutsche Forschungsgemeinschaft (DFG)