Technische Universität Berlin

FACHBEREICH 3
MATHEMATIK

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

by

FREDERIK STORK

No. 702/2000

Branch-and-Bound Algorithms for Stochastic Resource-Constrained Project Scheduling

Frederik Stork[*]

6. November 2000

Abstract

We study branch-and-bound algorithms for resource-constrained project scheduling where processing times of jobs are random. The objective is to find a so-called *scheduling policy* which minimizes the project makespan in expectation. The proposed procedures are based upon four classes of scheduling policies which differ considerably with respect to their computational tractability as well as with respect to the optimum costs that can be achieved within the respective class.

The purpose of the paper is twofold. First, we establish results on the trade-off between computational efficiency and solution quality for each of the considered classes of policies and evaluate their practical applicability for scheduling stochastic resource-constrained projects. 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.

1 Introduction

We consider the problem of scheduling jobs with uncertain processing times in order to minimize the project makespan in expectation. Precedence constraints have to be respected, that is, certain jobs must be completed before others can be executed. In addition, each job requires capacity of different resources while being processed, and the resource availability is limited. The problem generalizes the classical *resource-constrained project scheduling problem* which has been studies extensively in numerous variations; we refer to [3, 43] for reviews on different models and algorithms.

As has already been observed by Fulkerson [12], the assumption of random job processing times prevents the underestimation of project costs (that frequently occurs in real-world projects). However, surprisingly perhaps, only few attempts have been made to analyze resource-constrained project scheduling problems with random processing times. Most relevant for our study is the work of Igelmund and Radermacher [21, 20] which is briefly summarized below. Other contributions include

[*]Author's address: Technische Universität Berlin, Fachbereich Mathematik, Sekr. MA 6-1, Straße des 17. Juni 136, D-10623 Berlin, Germany. Email: stork@math.tu-berlin.de. Supported by the Deutsche Forschungsgemeinschaft (DFG) under grant Mo 446/3-3

meta heuristics [41, 42, 15], commercial software evaluation [9], and qualitative aspects [22]. Moreover, the NASA [19] has performed research on stochastic resource-constrained project scheduling in order to identify so-called *critical* jobs within space shuttle ground processing (there, jobs are called critical if their delay causes a delay of the entire project). The necessity of stochastic job processing times in resource-constrained project scheduling is probably best motivated by the following quotation taken from [19].

> "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." [19, Page 4]

Due to the combination of random job processing times and resource constraints, scheduling is usually done by so-called *policies*. A policy may be seen as an on-line decision process that defines which jobs are started at certain decision times t, based on the knowledge of the given distributions and the observed past up to t. Different classes of such policies have been considered in the literature, we refer to Möhring, Radermacher, and Weiss [29, 30] for a comprehensive characterization. Independently, Fernandez, Armacost, and Pet-Edwards [10] introduced scheduling policies for stochastic resource-constrained project scheduling; they interpret a policy as a multi-stage stochastic optimization problem, where each stage is identified with a decision time t. Our work is based upon so-called *preselective policies* that have been introduced by Igelmund and Radermacher [21, 20]. Roughly speaking, such policies define for each possible resource conflict among a subset F of jobs a *preselected* job $j \in F$ which is postponed if the corresponding resource conflict appears within the execution of the project. Igelmund and Radermacher discuss different properties of preselective policies which are of both theoretical and computational importance and develop a branch-and-bound algorithm in order to compute optimal preselective policies for stochastic resource-constrained project scheduling problems. To the best of our knowledge, this is the only reference in this direction. However, their computational experiments are limited to few small instances. Based on their work, we develop branch-and-bound procedures for preselective policies as well as for three previously studied subclasses thereof, so-called *ES-policies* [38], *linear preselective policies* [34], and *job-based priority policies*, e. g. [34], which are reviewed in Section 3 below. In contrast to Igelmund and Radermacher [20] and Radermacher [38], we enhance the procedures by utilizing recent algorithmic developments in the field of so-called AND/OR *precedence constraints* which allow a very efficient evaluation of preselective policies [33, 34]. In particular, we make use of an efficient algorithm to estimate the expected makespan of preselective policies and use a strong dominance rule that usually allows to prune large portions of the search trees.

The classes of policies we consider differ with respect to both their computational tractability and the optimum expected makespan that can be achieved within a class. We establish results on the trade-off between computational efficiency and solution quality for each of the considered classes of policies and evaluate their practical applicability for scheduling stochastic resource-constrained projects. 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 each of the other three classes of policies. We explore their computational efficiency on 1440 instances of different size that have been generated with the widely accepted instance generator ProGen [25].

The paper is organized as follows. After having formally described the considered scheduling model (Section 2) we review the considered classes of scheduling policies (Section 3). In Section 4 we describe the used branching schemes. We then state dominance rules for each of the branch-and-bound algorithms in Section 5. Section 6 gives a more detailed account of some ingredients that helped to speed up the computations. Our computational study is presented in Section 7, and we conclude with some remarks in Section 8.

2 Model and Notation

We are given a set $V = \{0, 1, \dots, n\}$ of jobs with random processing times that have to be scheduled non-preemptively such as to minimize the expected project makespan (that is, the time which is required to complete all jobs). However, unless explicitly stated, the proposed methods can as well be adapted to handle many other regular objective functions. We assume that job 0 is a dummy job that represents the project start. Its processing time is fixed to $p_0 = 0$. Processing times of the other jobs are given by a random vector $\boldsymbol{p} = (\boldsymbol{p}_1, \dots, \boldsymbol{p}_n)$ where $\boldsymbol{p}_i$ denotes the random processing time of job i. A particular sample of $\boldsymbol{p}$ is denoted by $p = (p_1, \dots, p_n) \in \mathbb{R}^n_+$. Precedence constraints are defined by pairs (i, j) of jobs with the meaning that job j cannot be started before job i has been completed. The set E_0 of precedence constraints defines a partial order relation on V which is denoted by $G_0 := (V, E_0)$. Resource constraints are given by a set of different, renewable resources K where for each $k \in K$ the value $R_k \in \mathbb{N}$ specifies the amount of available units of resource k. While in process, each job consumes an amount of $r_{jk} \leq R_k$, $r_{jk} \in \mathbb{N}$, units of resource $k \in K$. The jobs then have to be scheduled in such a way that at no time t the total resource consumption exceeds the resource availability. Resource constraints can alternatively be represented by a system $\mathcal{F} \subseteq 2^V$ of so-called *minimal forbidden sets*, i. e., inclusion-minimal sets F of pairwise not precedence-related jobs that cannot be scheduled simultaneously because they share some common limited resource. Note that, by definition, the number of minimal forbidden sets may grow exponentially in the number of jobs. An algorithm to compute $\mathcal{F}$ from the usual formulation of resource constraints by resource requirements and availability has been documented by Stork and Uetz [40].

If we assume that the distributions of job processing times are discrete, then the problem is a generalization of deterministic resource-constrained project scheduling. Consequently, if the encoding length of distributions polynomial in n, the problem to minimize the expected makespan is clearly NP-hard in the strong sense.

3 Scheduling Policies

General scheduling policies. Scheduling policies (or *strategies*) have been extensively studied and characterized by Möhring, Radermacher, and Weiss [29, 30], (see also Igelmund and Radermacher [21]). According to their definition, a scheduling policy defines for each decision time $t \geq 0$ a set $B \subseteq V$ of jobs that are started at t. Generally, policies may perform actions at any point in time, however, throughout the paper we assume that such *decision points* are $t = 0$ (project start) and job completions. Clearly, the jobs that are started at t must respect all precedence and resource constraints. That is, for each $j \in B$, all predecessors are completed by t and no minimal forbidden set is contained in the union of B and the set of jobs that are already in process at time t. Moreover, for the definition of B, a scheduling policy only uses information, that became available up to time t. This information includes the input data (set of jobs, precedence constraints, minimal forbidden sets, processing time distributions) and the processing and start times of jobs that completed by time t. In addition, information of the start times of the jobs which are in process at t can be exploited. However, the classes of policies we consider do not use the latter type of information. For a general definition of policies as well as a discussion of fundamental results related to stability, idleness, and the analytic behavior of policies we refer to Möhring, Radermacher, and Weiss [29, 30].

Once every job has been completed we know the processing times of the jobs and thus have a sample p from the random vector $\boldsymbol{p}$ of job processing times. Consequently, every policy Π may alternatively be interpreted as a function $\Pi : \mathbb{R}^n_+ \to \mathbb{R}^n_+$ that maps given samples of job processing times to vectors $S(p) \in \mathbb{R}^n_+$ 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 sample p by $S^\Pi_j(p)$ and its completion time by $C^\Pi_j(p) := S^\Pi_j(p) + p_j$. The corresponding random variables are denoted by $\boldsymbol{S}^\Pi_j$ and $\boldsymbol{C}^\Pi_j$, respectively. If no misinterpretation is possible we omit the policy superscript Π.

Probably the best-known class of scheduling policies is the class of *priority policies*. A priority policy orders all jobs according to a priority list and, at every decision time t, start as many jobs as possible in the order of that list. In deterministic (project) scheduling this is well-known as *Graham's List Scheduling* [16] (sometimes also called the *parallel list scheduling scheme*). However, priority policies have several drawbacks such as the fact that 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 increasing project duration due to decreasing job processing times, see [16]. Thus, if we think of a policy as a function that maps a sample of job processing times to feasible start times, priority policies are neither monotone nor continuous.

In the sequel we briefly review the classes of ES-policies, (linear) preselective policies, and job-based priority policies. Each such policy (in the view of a function) is monotone and continuous and consequently, Graham anomalies do not occur within project execution. Notice that we restrict the following synopsis to issues that are relevant for computational purposes; a detailed theoretical discussion of other topics is beyond the scope of the paper. For further details not mentioned here we refer to [29, 30, 28] as well as to Radermacher [38] (ES-policies), Igelmund and Radermacher [21, 20] (preselective policies), and Möhring and Stork [34] (linear preselective policies and job-based priority policies). In order to illustrate the presentation we use the following example which is taken from [21] (see also Figure 1).

Example 1. *Let* $G_0 = (V, E_0)$ *be given by* $V = \{1, 2, 3, 4, 5\}$ *and* $E_0 = \{(1, 4), (3, 5)\}$

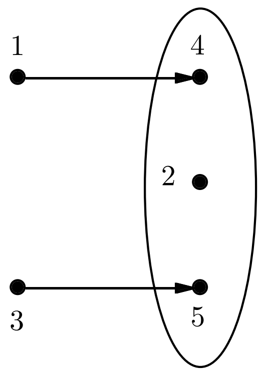

Figure 1: The precedence constraints of Example 1 together with a minimal forbidden set $\{2,4,5\}$ (to keep the figure simple we did not include the minimal forbidden sets $\{1,5\}$ and $\{2,3,4\}$).

and let the sets $\{1,5\}, \{2,3,4\}$ *and* $\{2,4,5\}$ *be minimal forbidden. 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.

Earliest Start policies (ES). A policy Π is called *Earliest Start policy* (*ES-policy*) if for each minimal forbidden set F there exists a pair (i,j), $i,j \in F$, $i \neq j$, such that for each sample p of job processing times, j is not started before i has been completed.

An ES-policies can be represented as an acyclic extension $G = (V,E)$ of the underlying precedence relation E_0 by adding all such pairs (i,j) to E_0. Then, in order to obtain a schedule $S(p)$ for a given sample p of job processing times we simply compute earliest job start times with respect to G, i. e., $S_0(p) := 0$ and

$$S_j(p) := \max_{(i,j)\in E} (S_i(p) + p_i) \qquad j \in V \setminus \{0\} \ . \tag{1}$$

Consider the ES-policy for Example 1 defined by $E = E_0 \cup \{(1,5),\,(2,3),\,(4,5)\}$. For the sample $p = E[\boldsymbol{p}]$ we obtain start times $S(p) = (0,0,5,3,8)$. Formula (1) immediately suggests that ES-policies viewed as functions $\Pi : p \to S(p)$ are monotone, continuous, and convex.

Preselective policies (PRS). A policy Π is called *preselective* if for each minimal forbidden set F there exists a job $j \in F$, such that for each sample p of job processing times, j is not started before some job $i \in F \setminus \{j\}$ has been completed. We call j a *waiting job* for F and define a *selection* to be a sequence $s = (s_1, \ldots, s_f)$, $s_\ell \in F_\ell$, of waiting jobs s_ℓ. Notice that, in contrast to ES-policies, the choice of i is not fix; it depends on the particular sample p.

A useful combinatorial representation of preselective policies — so-called *waiting conditions* — has been suggested in [34]. A waiting condition is given by a pair (X,j), $X \subset V$, $j \in V \setminus X$, where job j cannot be started before *at least one* job $i \in X$ has been completed. Each restriction induced by a minimal forbidden set F and its

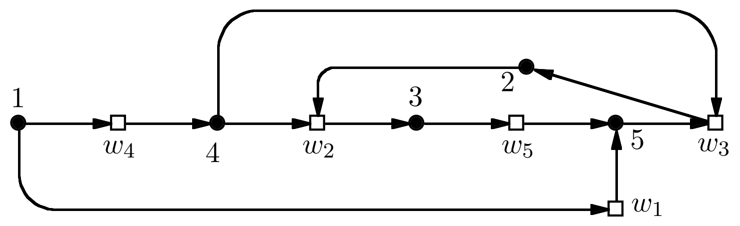

Figure 2: The digraph resulting from Example 1 and selection $s = (5, 3, 2)$. Circular nodes correspond to jobs while square nodes represent waiting conditions. Nodes w_1, w_2, and w_3 are induced by forbidden sets and the respective preselected job. Precedence constraints are represented by w_4 and w_5.

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 $(\{i\}, j)$. A set $\mathcal{W}$ of waiting conditions naturally 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) is connected to the node representing j. The set of waiting conditions which is induced by Example 1 with selection $s = (5, 3, 2)$ is $\mathcal{W} = \{w_1 = (\{1\}, 5)$, $w_2 = (\{2, 4\}, 3)$, $w_3 = (\{4, 5\}, 2)$, $w_4 = (\{1\}, 4)$, $w_5 = (\{3\}, 5)\}$. The associated digraph D representing $\mathcal{W}$ is depicted in Figure 2. In the scheduling literature, the concept of waiting conditions is also known as AND/OR precedence constraints, see, e. g., [13, 14, 33].

For a given selection s and resulting system of waiting conditions $\mathcal{W}$ we can construct a schedule for each possible sample $p \in \mathbb{R}^n_+$ by setting $S_0(p) := 0$ and

$$S_j(p) := \max_{(X,j)\in\mathcal{W}} \left(\min_{i\in X}(S_i(p) + p_i)\right) \qquad j \in V \setminus \{0\} \ . \tag{2}$$

A component-wise minimal solution to (2) can be computed by a generalization of Dijkstra's shortest path algorithm as has been described (partly independent from each other) by several authors; an early reference is Knuth [24]. We use a variation of this algorithm which has been proposed by Möhring, Skutella, and Stork [33]. This implementation runs in $\mathrm{O}(n + m + f \log f)$ time where $f := |\mathcal{F}|$ and m denotes the number of arcs in D. For algorithmic details on this algorithm we refer to [33]. Notice that there does not need to exist a finite solution $S(p) < \infty$ of (2). In this case the selection s is *infeasible* and hence does not define a preselective policy. In fact, if $S_j(p) = \infty$ for some arbitrary $p \in \mathbb{R}^n_+$ and $j \in V$ then this is the case for all possible samples. The selection $s = (1, 3, 2)$ for Example 1 is infeasible while the selection $s = (5, 3, 2)$ is feasible and for the sample $p = E[\boldsymbol{p}]$ Formula (2) yields $S(E[\boldsymbol{p}]) = (0, 8, 8, 3, 11)$. For details on the feasibility of sets of waiting conditions (and selections) we refer to [33, 34, 20].

Finally, a preselective policy as a function $\Pi : p \to S(p)$ is both monotone and continuous and thus does not show Graham anomalies. This is immediate with (2).

Linear preselective policies (LIN). The class of *linear preselective policies* is a subclass of the class of preselective policies. The underlying idea is to define the selection by a priority ordering L of the jobs in such a way that the preselected job of a minimal forbidden set F is the one with lowest priority, i. e., the last element of F in L. Note that we assume, without stating explicitly, that orderings of jobs are conform to the given precedence constraints E_0. Möhring and Stork [34] show that linear preselective policies have several nice properties which make them computationally more advantageous when compared to preselective policies. Most relevant is that the digraph D of waiting conditions which is induced by a linear preselective policy is acyclic. This allows earliest start times (defined by (2)) to be computed in $\mathrm{O}(n + m + f)$ time. Consider Example 1 and the linear preselective policy defined by the sequence $L = 3 < 1 < 5 < 2 < 4$. The resulting selection is $s = (5, 4, 4)$ and for the sample $p = E[\boldsymbol{p}]$ we obtain $S(p) = (0, 0, 0, 5, 3)$.

Since each linear preselective policy is also a preselective policy they inherit the analytic properties of being monotone and continuous.

Job-based priority policies (JBP). A policy Π is called *job-based priority policy* if it is linear preselective (according to some ordering L of jobs) and fulfills the additional restriction $S_i(p) \leq S_j(p)$ for each sample p and $i \prec_L j$. In order to compute earliest job start times for a given sample p we can start each job in the order of L as early as possible but not before the start time of some previously started job. A corresponding algorithm can be implemented to run in $\mathrm{O}(n|K| + |E_0| + n \log n)$ time and is thus not dependent on the number of minimal forbidden sets. In fact, job-based priority policies do not require the forbidden set representation of resource constraints at all. This particularly allows to apply such policies to very large projects where the (possibly exponential) number of minimal forbidden sets makes the use of forbidden set based policies computational inefficient. If the job-based priority policy $L = 3 < 1 < 5 < 2 < 4$ is applied to Example 1 and $p = E[\boldsymbol{p}]$ we obtain $S(p) = (0, 3, 0, 8, 3)$. Note that this schedule differs from the schedule obtained for a linear preselective policy (although the orderings of jobs which define the policies coincide). Like with the above defined classes of policies, if a job-based priority policy is viewed as a function, it is also monotone and continuous. In fact, if Π is a job-based priority policy for some 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 [34].

Finally, notice that job-based priority policies have previously been used within other (mostly deterministic) scheduling models where they occur in connection with, e. g., approximation algorithms [18, 31] and branch-and-bound procedures [39].

Optimum expected makespan. For a given sample p and associated completion times $C^{\Pi}(p)$ of jobs, $C_{max}(C^{\Pi}(p))$ denotes the makespan of the schedule resulting from p, and $E[C_{max}(\boldsymbol{C}^{\Pi})]$ denotes the expected makespan under policy Π. The objective is to minimize C_{max} *in expectation* over a class of policies. We therefore define a stochastic scheduling policy $\Pi^* \in \tau$ to be optimal with respect to some class τ of policies if Π^* minimizes the expected project makespan within τ, i. e., $E[C_{max}(\boldsymbol{C}^{\Pi^*})] = \inf\{E[C_{max}(\boldsymbol{C}^{\Pi})] | \Pi \in \tau\}$. We denote the optimum expected makespan that can be achieved within a class τ of polices by ρ^{τ}.

If processing times are deterministic it is not hard to see that each of the above defined classes of policies achieves the optimum value, i. e., $\rho^{\text{PRS}} = \rho^{\text{LIN}} = \rho^{\text{ES}} = \rho^{\text{JBP}}$.

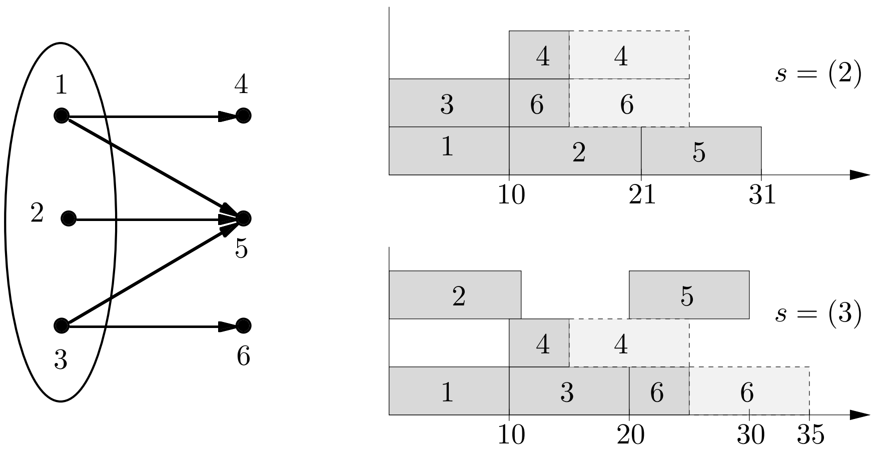

Figure 3: The instance given in Example 2 together with Gantt charts for the selections $s = (2)$ and $s = (3)$ (the selection $s = (1)$ is symmetric to $s = (3)$). Both Gantt charts show the different processing times of the jobs 4 and 6. While $E[C_{max}(\boldsymbol{p})] = 31$ for $s = (2)$ we obtain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

In the stochastic case, however, the behavior is quite different. By definition we immediately obtain $\rho^{\text{PRS}} \leq \rho^{\text{LIN}} \leq \rho^{\text{JBP}}$. Moreover, each ES-policy can easily be transformed into a preselective policy with no larger makespan and consequently, $\rho^{\text{PRS}} \leq \rho^{\text{ES}}$. On the other hand, it can verified by simple counter-examples that the classes ES and JBP are not comparable in terms of their optimum value. The same is true for the classes ES and LIN: It is easy to construct instances with $\rho^{\text{LIN}} < \rho^{\text{ES}}$; an instance with $\rho^{\text{LIN}} > \rho^{\text{ES}}$ is given in [34, Example 2]. Notice that all quoted inequalities become strict for appropriately constructed instances. An analysis of Example 1 yields the ordering $\rho^{\text{PRS}} = \rho^{\text{LIN}} < \rho^{\text{ES}} < \rho^{\text{JBP}}$. In fact, as will be shown in the computational study in Section 7.3 below, we obtain the same ordering for the entire test set we used within our computational experiments.

4 Branch-and-Bound and Random Processing Times

In this section we recapitulate two branching schemes that have been previously applied to resource-constrained project scheduling problems and discuss their applicability in the stochastic environment. Moreover, we describe how the expected project makespan of a given policy can be (approximately) computed.

Before, 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 2. *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 forbidden. 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 lead to 4 different samples. 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 3). 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. We make use of two different branching schemes that have previously been proposed in the literature, namely 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*. Each node v in the 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 $\{F_1, \dots, F_{d(v)}\}$ of minimal forbidden sets (for preselective policies, this equals a partial selection $s = (s_1, \dots, s_{d(v)})$). Moreover, notice that within the enumeration of linear preselective policies, a node v is discarded if the partial selection s induced by v cannot be defined by an ordering L of the jobs which extends E_0. This can be tested in $\mathrm{O}(n + m + f)$ time by checking whether the digraph D of waiting conditions which result from G_0 and s is acyclic [34]. Thus, for ES-policies the number of children of a node v is quadratic in the number of jobs contained in the associated minimal forbidden set whereas the number of children is linear in the preselective and the linear preselective case. 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 as well. However, for ES-policies, the maximal depth can be bounded by $n(n-1)/2$ (see Section 5.1 below).

Notice that, in deterministic resource-constrained project scheduling, a variation of the forbidden set branching scheme turned out to be very successful. There, each node represents an infeasible schedule in which a resource conflict is systematically resolved by so-called *minimal delaying alternatives*, see, e. g., [5, 6, 11]. However, this branching scheme is not applicable to the stochastic case because it makes excessive use of deterministic job processing times.

We next describe the *precedence-tree branching scheme* which is described in, e. g., Patterson, Słowiński, Talbot, and Węglarz [35]. Here, linear extensions of the underlying precedence constraints 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 $i \prec_L j$ for all $(i, j) \in E_0$ (notice that we sometimes view L as a set of jobs). We call such orderings *initial segments* of G_0. 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 i 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. So far, the precedence-tree branching scheme has been applied to deterministic resource-constrained project scheduling problems only, a recent reference is [39].

Computing the expected makespan. For a given node in the search tree, lower bounds on the expected makespan are computed in order to discard the node from further consideration. (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*. Unfortunately, already for precedence-constrained jobs without any resource restrictions, the computation of the expected length of a critical path is #P-complete if each job processing time distribution has two (discrete) values (see Hagstrom [17]). Although for complex distributions the problem has a longer encoding which could possibly admit a polynomial algorithm, such an efficient algorithm seems very unlikely to exist, since Hagstrom has also shown that — unless P=NP — the problem cannot be solved in time polynomial in the number of possible values of the makespan (assuming discrete distributions). For these reasons one usually approximates the expected makespan of a given policy by *simulation* of the given distributions of job processing times. We then obtain a set P of samples for $\boldsymbol{p}$, that can be used to approximately calculate $E[C_{max}(\boldsymbol{C}^{\Pi})]$. This is done by computing the average makespan of the schedules resulting from these samples, i. e., $E[C_{max}(\boldsymbol{C}^{\Pi})] \approx \frac{1}{|P|} \sum_{p \in P} C_{max}(C^{\Pi}(p))$. The usage of simulation techniques for stochastic, discrete optimization problems has been discussed by Kleywegt and Shapiro [23] who argue (among other issues) that for $|P| \to \infty$ the optimum value that results from the simulated data converges to the optimum value of the original instance.

5 Dominance Rules

This section is concerned with several so-called *dominance rules*. For a given node v in some search tree and a sample $p \in \mathbb{R}^n_+$, denote by $C_{max}(v, p)$ the makespan of a node with smallest makespan in the subtree rooted at v. We call another node u *dominated* by v if $C_{max}(v, p) \leq C_{max}(u, p)$ for all samples $p \in \mathbb{R}^n_+$. A dominance rule is a method that (usually heuristically) identifies dominated nodes. If this is the case, clearly, we can disregard u from further consideration. Notice that it must be taken care of what might be called *cross pruning*, that is, nodes that dominate each other must not all be removed from the search. Finally, in accordance to the above definition, we call a policy Π *dominated* if there exists another policy $\Pi' \neq \Pi$ such that $S^{\Pi}(p) \geq S^{\Pi'}(p)$ for all samples $p \in \mathbb{R}^n_+$.

5.1 Earliest Start Policies

In the forbidden set branching scheme all dominance rules we propose are based on the same principle. Let v be a node of the search tree. The decisions that have been made at the ancestors of v may implicitly solve resource conflicts that occur due to minimal forbidden sets that have not been considered for branching so far. We call such forbidden sets *implicitly resolved*.

For ES-policies, a dominance rule based on this principle is easily obtained [38]: Each node v in the search tree can be identified with a set of new precedence constraints which are added to E_0. These constraints together with the resulting transitive constraints must be respected in each node which is located in the subtree rooted at v. Denote the new set of ordered pairs by E and suppose that F is the next minimal forbidden set, on which branching is performed. Then, if we have $(i, j) \in E$ for jobs $i, j \in F$, the child node u of v in which F is resolved by postponing j until i has been completed, dominates all other child nodes $w \neq u$ of v. This observation is immediate with the following lemma which also shows that the dominance criterion is both necessary and sufficient.

Lemma 1. *(Radermacher [37]) Let (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 $E' \subseteq E$.*

With respect to an implementation of a dominance rule resulting from the above discussion, Radermacher [38] and Bartusch [2] 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 $\mathrm{O}(n^2 f)$ space, which is not acceptable for a large number of minimal forbidden sets. Instead, we implemented an algorithm which does not need the memory-expensive destruction matrix. The algorithm also works for preselective policies and is described in the following Section 5.2.

Finally, notice that if the above described dominance rule is employed, we obtain the following property of the branch-and-bound tree. On any path from the root to some leaf, the number of nodes with more than one child node is bounded by $n(n-1)/2$.

5.2 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 $s = (s_1, \ldots, s_{d(v)})$. Together with the initial set E_0 of precedence constraints, s defines a set $\mathcal{W}$ of waiting conditions. Like with traditional precedence constraints, a sets of waiting conditions may imply other "transitive" waiting conditions $(X, j) \notin \mathcal{W}$. That is, for each sample p of job processing times $\mathcal{W}$ guarantees that $S_j(p) \geq \min_{i \in X}(S_i(p) + p_i)$. Consequently, if some waiting condition (X, j) is implied by $\mathcal{W}$ and for some minimal forbidden set

F, $X \subset F$ and $j \in F$, then F is implicitly resolved with respect to s. Therefore, if we branch over F in node v, it suffices to only consider the branch u which represents the choice of j as the preselected job for F; all other child nodes are dominated by u. The following lemma which is straightforwardly obtained from [34, Lemma 5.2], establishes the correctness of the above argumentation.

Lemma 2. *(Möhring and Stork [34]) Let $\mathcal{W}$ and $\mathcal{W}'$ be sets of waiting conditions that represent preselective polices Π and Π', $\Pi \neq \Pi'$, respectively. Then, Π is dominated by Π' if and only if each $(X, j) \in \mathcal{W}'$ is implied by $\mathcal{W}$.*

Notice that Lemma 2 can be seen as a generalization of Lemma 1 by considering waiting conditions instead of traditional precedence constraints.

Based on the above dominance rule, Möhring, Skutella, and Stork [33] have devised an algorithm to verify whether a given minimal forbidden set is implicitly resolved. The underlying idea is best described in terms of waiting conditions, so suppose that we are given a set $\mathcal{W}$ of waiting conditions which is associated with some node of the search tree and let F be the minimal forbidden set for which we want to test whether it is implicitly resolved. We then proceed by constructing a subset L of the jobs that can be started for all samples $p \in \mathbb{R}^n_+$ without the restriction that some job from F must have been completed previously. This is done in the following greedy way: while there exists a job $i \in V \setminus F$ that is not a waiting job of any of the waiting conditions in $\mathcal{W}$, i is inserted into L. Whenever a waiting condition (X, j) becomes satisfied (which is the case when some $i \in X$ is being added to L), (X, j) is deleted from $\mathcal{W}$. Consider then the jobs from F after the recurrence has terminated. Möhring, Skutella, and Stork [33, Corollary 6] show that each job $j \in F$ that cannot be added to L without violating any of the remaining waiting conditions is a waiting job for F. For further theoretical details on the dominance of preselective policies and the correctness of the above outlined algorithm we refer to [34] and [33].

We have implemented the following variation of the above 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 there exists some implicit waiting job $j \in F$. If this is the case, we label F to be implicitly resolved 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 (in the case of ES-policies the running time reduces to O$(n+|E|)$). 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.

5.3 Linear Preselective Policies via 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, the integration of the dominance rule exposes a characteristic of linear preselective policies which requires explanation. Suppose that some minimal forbidden set F is implicitly resolved with respect to some partial selection s, say, because the waiting condition $(F \setminus \{j\}, j)$ is implied. According to the dominance rule we then label F to

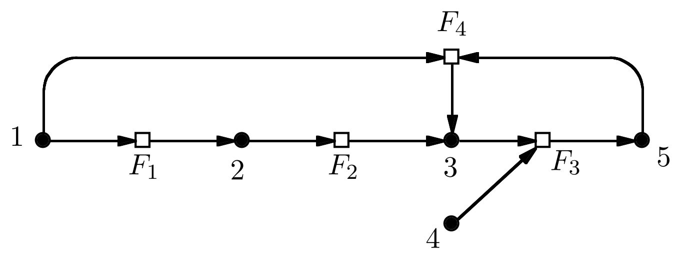

Figure 4: The figure shows the (cyclic) digraph of waiting conditions which results from Example 4 and selection $s = (2, 3, 5, 3)$.

be implicitly resolved. Hence we do not need to perform a branching on F. However, if j is explicitly preselected with respect to F then the resulting (partial) selection s' needs not necessarily be linear preselective. We demonstrate this observation by the following example (see also [34]).

Example 3. *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.*

Consider the (linear preselective) partial selection $s = (2, 3, 5)$; the resulting digraph of waiting conditions is given in Figure 4. 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 a superfluous waiting condition $(\{1, 3\}, 5)$.

We obtain the following consequence for the algorithm to enumerate linear preselective policies by the forbidden set branching scheme: If we employ the dominance rule as proposed in Section 5.2, the algorithm may output a policy that has a smaller optimum value than all linear preselective policies. However, for our computational study, we integrated the dominance rule into the algorithm. The decision is based on the following two facts. First, any selection s that is obtained from the branch-and-bound algorithm can still be represented by an acyclic network of waiting conditions. In particular, s always induces a linear preselective policy for an instance I' which is "tighter" than the input instance I in the sense that I' is obtained from I by removing jobs from minimal forbidden sets. For example, if I is the instance given in Example 3, then $s = (2, 3, 5, 3)$ is not linear preselective for I but s is linear preselective for the instance I' where $F_4 = \{1, 3, 5\}$ is replaced by the minimal forbidden set $\{1, 3\}$. Second, suppose that we only disregard 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. 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 5.2 does not yield an optimum expected makespan that is smaller than the value obtained without the dominance rule.

5.4 Linear Preselective Policies via Precedence-Tree Enumeration

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 some jobs $V \setminus L$. This clearly is 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' 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 $i <_{id} j$ 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 $i <_{id} j$).

In the sequel we describe a simple transformation rule that constructs 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 remove 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 removed. Consider now some job i in the ordering L. In order to apply the transformation rule we traverse L backwards until a predecessor $j \in Pred(i)$ or some job j with $j >_{id} i$ is found. Notice that $j >_{id} i$ and $j <_L i$ implies that $j \notin Pred(i)$. In the first case 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 $i >_L h \geq_L j$. If this is the case, it is easy to see that 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. Hence, we discard L from further consideration.

Lemma 3. *Let s be a selection and let L^* be the lexicographically smallest ordering which represents s. The above transformation rule deletes all orderings $L >_{lex} L^*$ from the search tree that yield selection s.*

Proof. Suppose that $L >_{lex} L^*$ is not deleted 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 $i <_{id} j$, $i >_L j$ and that $Pred(i) \cap B = \emptyset$, where B is the set of jobs $i >_L h \geq_L j$. Moreover, since L and L^* define the same selection s there exists no minimal forbidden set F_ℓ with $s_\ell = i$ and $F_\ell \cap B \neq \emptyset$. Consequently, at the point of branching where i is appended to L, the transformation rule identifies the ordering L' which is obtained from L by moving i to the position directly before j. Since $L' <_{lex} L$, L is deleted — a contradiction. □

We employ the above result by defining i to be the last job in the currently considered initial segment L and the perform the above transformation rule. Moreover, with the following additional feature we are able to remove some of the dominated orderings already at the branching process. Here, $\mathcal{F}_j$ denotes the set of minimal forbidden sets F with $j \in F$.

Lemma 4. *For a given initial segment L let B denote the set of jobs i that can be appended to L without violating the precedence constraints (that is, $i \in B$ if $Pred(i) \subseteq L$). If B contains some job j such that $(F \setminus \{j\}) \subseteq L$ for all $F \in \mathcal{F}_j$ then all extensions of L by one of the jobs $i \in B \setminus \{j\}$ are dominated.*

Loosely speaking, if neither the addition of j to L nor the addition of j to some extension of L yields a new preselected job in the associated selection then branching at L is not necessary. It obviously suffices to only consider the branch where j is appended to L.

Notice that, if the above described dominance rules are used simultaneously, due to cross-pruning, we may cut off all optimal solutions. We therefore only use the following restricted version of Lemma 4. Let L, B, and $j \in B$ be as defined in Lemma 4. Instead of cutting off all extensions of L by some $i \in B \setminus \{j\}$, we only remove those branches where the job i which is to be appended to L fulfills $i >_{id} j$. Consequently, we never lose the lexicographically smallest of these initial segments.

5.5 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 the algorithm for job-based priority policies as outlined in Section 3. Sprecher [39] 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.

Lemma 5. *Let L be an ordering of the jobs and let i and j be jobs with the properties $j <_L i$ 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 $B \subseteq \{h | h <_L i\}$ be the set of jobs that can run simultaneously to i with respect to L'. Then L (as a job-based priority policy) is dominated by L' if i is contained in no minimal forbidden set F with $F \subseteq B$.*

Proof. Notice that, since L is feasible and $Pred(i) \subseteq Pred(j)$ we have that i is not related to any job h with $i >_L h \geq_L j$. Consequently, L' respects the precedence constraints. We show that $S'_h(p) \leq S_h(p)$ for all samples $p \in \mathbb{R}^n_+$ and all $h \in L$ (S and S' denote the schedules resulting from L and L', respectively). For $h <_L j$ this is trivial, since $S'_h(p) = S_h(p)$. For i we even obtain $S'_i(p) \leq S_j(p) (\leq S_i(p))$. This follows from $Pred(i) \subseteq Pred(j)$ and the fact that there is no forbidden set among i and the jobs which are in process at time $S_j(p)$ with respect to L'. By essentially the same argumentation we have that $S'_h(p) = S_h(p)$ for all $i >_L h \geq_L j$. Finally, recall that the orderings L and L' on the remaining jobs $h >_L i$ coincide, hence $S'_h(p) \leq S_h(p)$ is also valid for these jobs. $\square$

Recall that the reason why we consider job-based priority policies is to avoid the handling of exponentially many minimal forbidden sets. Consequently, we lose some structural information which we can access within the computation of optimum (linear) preselective policies and ES-polices. In particular, we cannot decide efficiently whether a given job i is contained in some minimal forbidden set or not, see Stork and Uetz [40]. However, 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 (see Stork and Uetz [40] and Möhring [27]). The algorithm involves two maximum flow computations in an auxiliary network of size $\mathrm{O}(n + |E_0|)$. We incorporate the result of Lemma 5 as follows. For each node in the search tree with associated initial segment L of some jobs we assume job i to 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). To avoid cross pruning, we additionally require that $j >_{id} i$. We then compute B as the set of jobs $h <_L i$ without successors in the set $\{i\} \cup \{\ell | \ell <_L j\}$. Since a job cannot be started earlier than the previously scheduled jobs, the jobs $L \setminus B$ cannot run simultaneously to i with respect to L'. Finally, we test whether there is a resource conflict between i and some of the jobs in B by the algorithm as suggested in [27]. If no conflict exists, L is discarded, otherwise we stop the dominance test.

6 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 7.4 below.

6.1 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*. They clearly require fixed 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 and sort the jobs with respect to non-decreasing start times S_j. From the resulting ordering L of the jobs (which clearly respects the precedence constraints) we directly obtain 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.

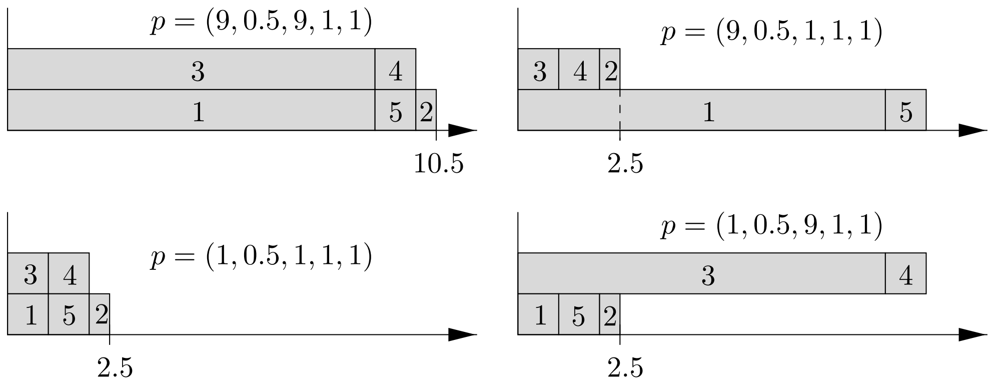

Figure 5: The figure shows a Gantt chart for each of the four possible samples resulting from Example 4. The instance itself (precedence constraints, minimal forbidden set) is depicted in Figure 1.

6.2 Critical Path Lower Bound and Jensen's Inequality

Since the computation of the critical path lower bound is time consuming in the stochastic case (see Section 4, last paragraph) we additionally employ a variation of this bound which can be computed much faster. The bound is based on Jensen's inequality and is applied before the expensive computation of the critical path lower bound. 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 4. *Let $G_0 = (V, E_0)$ be given by $V := \{1, 2, 3, 4, 5\}$ and $E := \{(1, 4), (3, 5)\}$ 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 4 and 6 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 processing times we have $S_2(E[\boldsymbol{p}]) = 6$ while $E[\boldsymbol{S}_2] = 4$ (see Figure 5). Note that even the expected *completion* time $E[\boldsymbol{C}_2] = 4.5$ is less than the minimum of the expected *start* times of the other jobs in the minimal forbidden set ($E[\boldsymbol{S}_4] = E[\boldsymbol{S}_5] = 5$).

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 samples p. As for ES-policies, each convex computation is bounded by Jensen's inequality. If we adapt (2) by setting $S'_0(p) := 0$ and

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

for all $j \in V \setminus \{0\}$ we obtain the following lemma.

Lemma 6. $S'_j \leq E[\boldsymbol{S}_j]$ *for all jobs* $j \in V$.

The lemma follows directly from Jensen's inequality and the monotonicity of preselective policies. As a consequence, if $C'_j := S'_j + E[\boldsymbol{p}_j]$, $C_{max}(C')$ is a lower bound for $E[C_{max}(\boldsymbol{C})]$ 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 [7] (see also Arnold [1]), however, the results are of the same order of magnitude as the lower bound based on Lemma 6. We therefore did not include such bounds into our experiments.

6.3 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., [26, 39]. Let $\boldsymbol{h}_j$ be the random variable of the *head* of job $j \in W$, that is, a lower bound on the expected start time $\boldsymbol{S}_j$ (e. g., the length of a longest chain in the partially ordered set that is induced by the predecessors of j in G_0). Moreover, let $\boldsymbol{t}_j$ denote the random variable of the *tail* of job $j \in W$, i. e., the length of a longest chain in the partially ordered set that is induced by the successors of j in G_0.

Lemma 7. *Let* $W \subseteq V$ *be a subset of jobs that can pairwise not be scheduled in parallel. Then* $\min_{j\in W}(E[\boldsymbol{h}_j]) + \sum_{j\in W} E[\boldsymbol{p}_j] + \min_{j\in W}(E[\boldsymbol{t}_j])$ *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, $E[\boldsymbol{h}_i] + \sum_{h\in W} E[\boldsymbol{p}_h] + E[\boldsymbol{t}_j]$ 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. $\square$

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 the jobs $W = \{3, 4\}$ form a minimal forbidden set, i. e., they must be scheduled sequentially. Then the formula of Lemma 7 yields a "lower bound" of 9 while the expected makespan of any priority policy is 8.5 (for a definition of priority policies see Section 3).

We have included the following implementation into our experiments. As a preprocessing step we compute different sets W of jobs and their deterministic tail. The sets are obtained 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 $\mathrm{O}(n^2)$ 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. The resulting single machine instance with heads and tails is fed into an algorithm as proposed by Carlier [4, 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 bound. The algorithm computes the best bound that can be achieved from any subset of W by a preemptive relaxation in $\mathrm{O}(|W| \log |W|)$ time.

6.4 Sorting the Forbidden Sets

In general, an important ingredient of branch-and-bound algorithms 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 usually 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 scenario where F is selected for the first branching. For each such scenario we compute the number of branches b 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. As a tie-breaker we choose the average increase of the global lower bound taken over all branches that result from a single scenario. Notice that, if none of the branches can be pruned by lower bound computation, the minimal forbidden sets are ordered by increasing cardinality.

6.5 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 enumeration 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 bound on the expected makespan and the depth of that node in the search tree.

7 Computational Study

7.1 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 total we report on five algorithms, that are, the computation of optimum preselective policies and ES-policies (both forbidden set branching scheme), job-based priority policies (precedence tree branching scheme), and two variations of linear preselective policies (forbidden set and precedence tree branching scheme). We abbreviate the five algorithms by PRS-FS, ES-FS, JBP-PT, LIN-FS, and LIN-PT, respectively.

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 6.1) as well as the lower bounds described in Sections 6.2 and 6.3 are switched on. Furthermore we employ the search strategy as described in Section 6.5 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 5. 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. The default type of the distribution of job processing times is a Gamma distribution with a variance of 3. Finally, we generate 200 samples out of the distributions, which turned out to provide a reasonable tradeoff between the precision of the expected makespan on the one hand and on 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 7.4 below.

7.2 The Test Sets

We have applied our algorithms to a test set which is created by the widely accepted instance generator ProGen [25]. 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 other, previously created ProGen test sets, the instances have been generated by modifying three parameters of the instance generator, the *network complexity* (NC) which reflects the average number of direct successors of a job, the *resource factor* (RF) which describes the average number of resources required in order to process a job divided by the number of resource types, and the *resource strength* (RS), which is a measure of the scarcity of the resources. 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. For further details on the instance

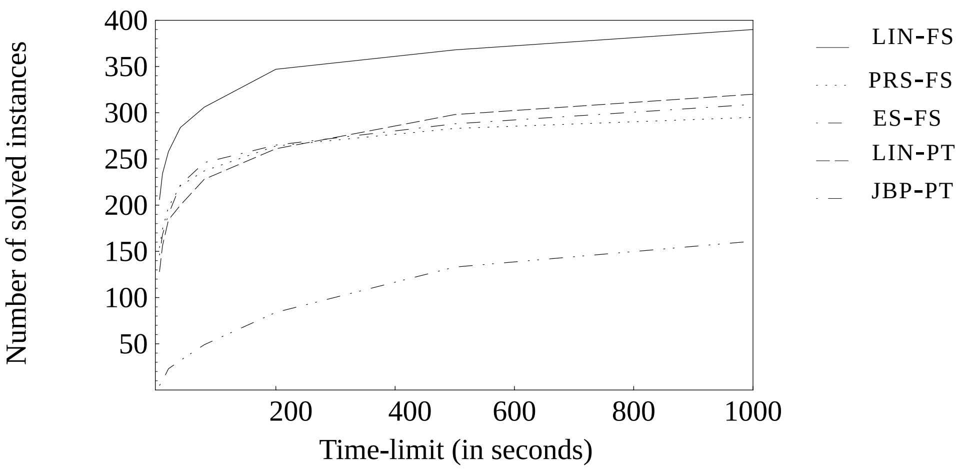

Figure 6: The number of optimally solved instances (out of 480 instances with 20 jobs each) depending on given time-limits. The ordering for the time limit of 1000 seconds is as follows (from top to bottom): LIN-FS, LIN-PT, ES-FS, PRS-FS, JBP-PT.

generator we refer to [25]. In Section 7.5 we also report on instances with 30 and 60 jobs, respectively. Details on the instances we consider are given in Section 7.5.

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

7.3 Comparison of the Procedures

Performance of the different procedures. We start the study by reporting on the computational expenses that are required by the different algorithms. Figure 6 shows for each of the five algorithms how many of the 480 instances can be solved optimally for different time limits. The plot shows 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 that 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 5.5 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

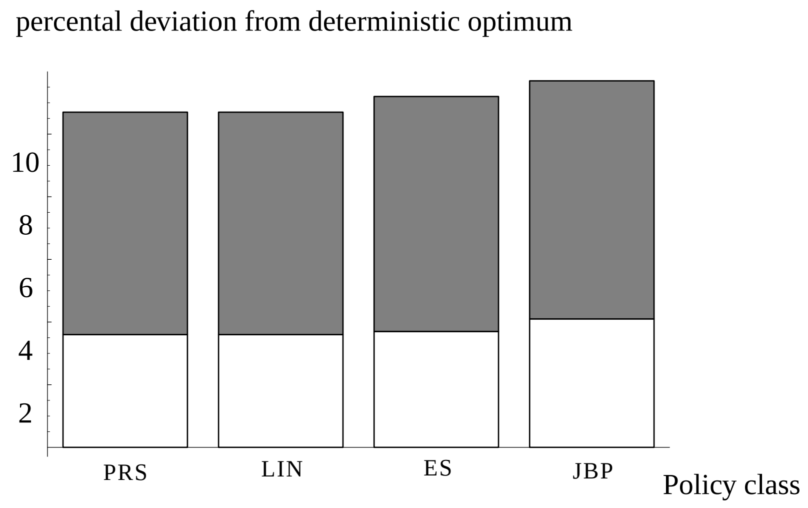

Figure 7: Average and maximum percental deviation of the expected makespan from the deterministic optimum. The figures are based upon 107 instances that were solved optimally by all algorithms.

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-policies, 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 O(n^2) compared to O(n) for the other procedures (see Section 4).

Comparison of optimum costs. We next discuss how the different values of the optimal expected makespan of the considered classes of policies are related to each other (recall that these values may differ considerably). Figure 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 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 7 also exposes the underestimation error [12] within deterministic planning. We see that, on average, the expected makespan is more than 4% larger than the deterministic makespan (with respect to processing times $E[\boldsymbol{p}]$). Even more, the maximal percental deviation is occasionally greater than 10%.

	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	2.8	100	162
JBP-PT	161	70	57	394

Table 1: The impact of the dominance rule as described in Section 5. The average running 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).

7.4 Impact of additional ingredients

In this section we test how the dominance rules as well as the various ingredients as proposed in Section 6 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 1 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 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 is displayed in Table 2. 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 based bound 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 based bound is switched off (recall that the single machine relaxation requires the output of the critical path based bound). For each variation of the parameters we show the number of solved instances as well as the average running 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%

	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 2: The impact of the lower bounds as described in Section 6. The average running 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 3: The impact of the search strategy as described in Section 6. The average running 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).

(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 6.2) 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 3. 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 running 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 Section 6.5 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

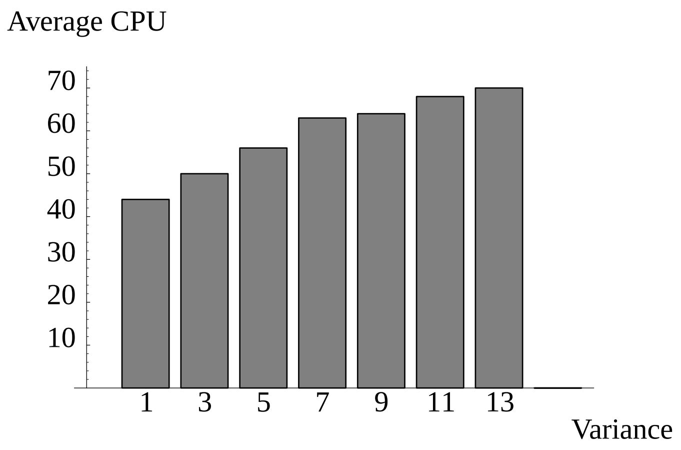

Figure 8: Dependency of the running time (in seconds) on the chosen variance of the distributions. The figures are based upon 366 instances that were solved optimally for each variance ($\{1, 3, 5, 7, 9, 11, 13\}$) for processing times. The used type of distribution was the Gamma distribution.

that we only document the results obtained for LIN-FS, since the behavior of each of the other algorithms is equivalent (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 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 running time increases. Figure 8 displays the dependency of the running 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 with a variance of 13.

Impact of the number of samples. The number of samples 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 sample. In Figure 9 we show for different numbers of samples the average and maximum percental deviation of the expected makespan from the deterministic problem with $p = E[\boldsymbol{p}]$. For 200 samples – which we have chosen as default – the expected makespan varies only little when compared to larger sampling sizes. On the other hand, the running time drastically increases with the number of samples. The average running times depending on the number of samples is displayed in Figure 10.

Sensitivity of the objective 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

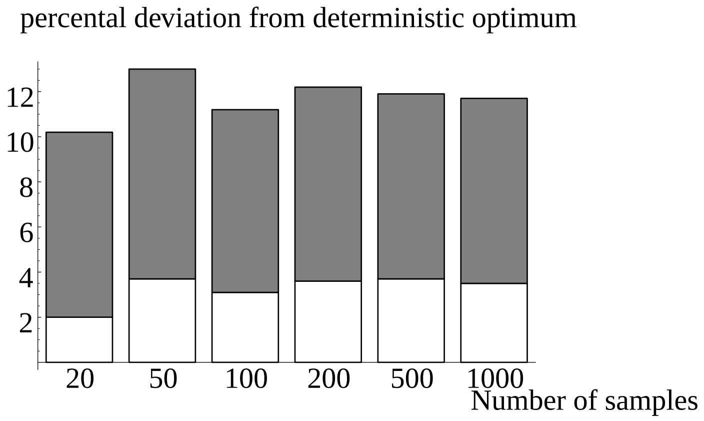

Figure 9: Average and maximum percental deviation of the expected makespan from the deterministic optimum (obtained with LIN-FS). The figures are based upon 344 instances that were solved optimally by all variations of the number of samples ($|P| \in \{20, 50, 100, 200, 500, 1000\}$).

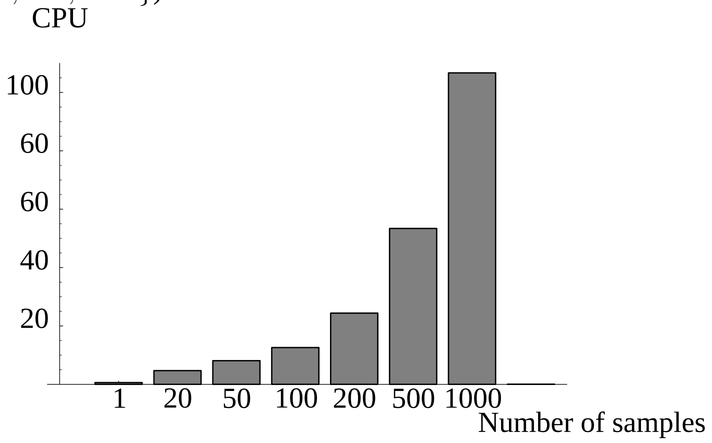

Figure 10: Average running times (in seconds) depending on the number of samples (obtained with LIN-FS). The figures are based upon 344 instances that were solved optimally by all variations of the number of samples ($|P| \in \{1, 20, 50, 100, 200, 500, 1000\}$).

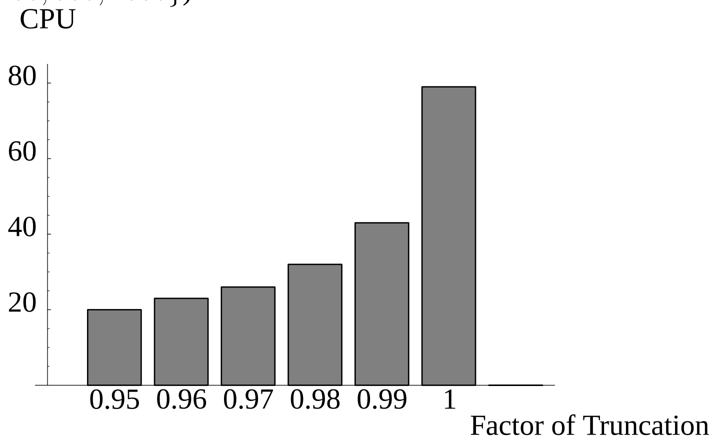

Figure 11: The running time (in seconds) for different values of truncations, averaged over 390 instances that can be "optimized" by all variations of truncation ($\alpha \in \{1.00, 0.99, 0.98, 0.97, 0.96, 0.95\}$).

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 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 running 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. 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 objective 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 objective function *expected makespan* to the deterministic counterpart.

Example 5. *Consider Example 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 samples. Suppose that the selection $s = (5, 4, 4)$ is determined by some constructive heuristic and serves as an initial upper bound which is 20 for the deterministic problem with expected processing times 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). 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 running 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 running 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.

7.5 Application to Larger 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 been taken from the PSPLIB [36] and were created by ProGen [25] (like the test set with 20 jobs that we used in the previous sections). The parameter setting for generating the instances is exactly as prescribed in Section 7.2, hence, in total, there are 480 instances with 30

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	77	2	394	7
JBP-PT	60	0	0	480	0

Table 4: 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 2), memory and time limit within the branch-and-bound (Columns 3 and 4), and the number of optimally solved instances (Column 5). We restricted the running time to 100 seconds and the memory limit to 50 MB per instance.

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 [15]. For deterministic processing times the instances with 30 jobs can be solved to optimality by tailored branch-and-bound procedures [6, 39, 8], however, for instances with 60 jobs, even these procedures often fail to compute optimal solutions within acceptable computation time.

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 [36] are displayed in Table 4. The second column gives 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 50MB. The figures in Columns 3 and 4 display the number of instances where the branch-and-bound was aborted due to the memory and time limit, respectively. The fifth column finally gives the number of instances that were solved to optimality. For instances with 30 jobs, although the number of minimal forbidden sets 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 $\rho^{\text{PRS}} \leq \rho^{\text{LIN}}$). 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 (note that these figures are based on 200 samples). 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 [15]. 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 [15]. 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 and Gonik [15] compute a feasible solution for that instance with expected makespan 448 (we rounded all reported values appropriately). At each job completion time t, 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 all jobs B that are precedence-feasible at t, a subset $B' \subseteq B$ of jobs is started at t with the property that $\sum_{j\in B'} r_{jk} \leq R_k$ for all k and $\sum_{j\in B'} q_j$ is maximized. Notice that, for $|K| = 1$, this is a $\{0,1\}$-Knapsack problem, hence, to obtain the above solution, Golenko and Gonik solved an NP-hard problem at each job completion. They also suggest to heuristically compute the set B' which resulted in a solution of (rounded) 461. They do not report on running times of the heuristics. In fact, already the starting solutions of our algorithms (see Section 6.1) 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 to compute feasible solutions of good quality.

8 Concluding Remarks

We have presented a computational evaluation of different branch-and-bound procedures for computing optimal scheduling polices for stochastic resource-constrained project scheduling problems. The experiments clearly demonstrate that linear preselective polices are much better computationally 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 based on the forbidden set branching scheme. If a large number of minimal forbidden sets prevents the use of this approach the only remaining alternative (among the classes of policies considered in the paper) is to perform the project according to a job-based priority policy. We have demonstrated that the optimum makespan among this class of policies is only slightly larger when compared to optimum makespan within the class of preselective policies. Hence, job-based priority policies are a good choice for local search heuristics which will be part of future research.

Acknowledgment. The author is grateful to Karin Hecht who has implemented parts of the code.

References

[1] B. C. Arnold. Bounds on the expected maximum. *Communications in Statistics, Theory and Methods*, 17:2135–2150, 1988.

[2] M. Bartusch. *Optimierung von Netzplänen mit Anordnungsbeziehungen bei knappen Betriebsmitteln*. PhD thesis, Rheinisch-Westfälische Technische Hochschule Aachen, 1984.

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

[4] J. Carlier. The one–machine sequencing problem. *European Journal of Operational Research*, 11:42–47, 1982.

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

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

[7] L. P. Devroye. Inequalities for the completion times of stochastic PERT networks. *Mathematics of Operations Research*, 4:441–447, 1979.

[8] U. Dorndorf, E. Pesch, and T. Phan Huy. A branch-and-bound algorithm for the resource-constrained project scheduling problem. Preprint, Institut für Gesellschafts- und Wirtschaftswissenschaften, Universität Bonn, Germany, 1999.

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

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

[11] A. Fest, R. H. Möhring, F. Stork, and M. Uetz. 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, 1998. Revised 1999.

[12] D. R. Fulkerson. Expected critical path length in PERT networks. *Operations Research*, 10:808–817, 1962.

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

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

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

[16] R. L. Graham. Bounds on multiprocessing timing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[17] J. N. Hagstrom. Computational complexity of PERT problems. *Networks*, 18:139–147, 1988.

[18] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22:513–544, 1997.

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

[20] G. Igelmund and F. J. Radermacher. Algorithmic approaches to preselective strategies for stochastic scheduling problems. *Networks*, 13:29–48, 1983.

[21] G. Igelmund and F. J. Radermacher. Preselective strategies for the optimization of stochastic project networks under resource constraints. *Networks*, 13:1–28, 1983.

[22] T. Jørgensen and S. Wallace. The stochastic resource-constrained project scheduling problem: Qualitative properties. Technical report, Section of Managerial Economics and Operations Research, Department of Industrial Economics and Technology Management, Norwegian University of Science and Technology, Trondheim, Norway, 1999.

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

[24] D. E. Knuth. A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6:1–5, 1977.

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

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

[27] R. H. Möhring. Algorithmic aspects of comparability graphs and interval graphs. In Ivan Rival, editor, *Graphs and Order*, pages 41–101. D. Reidel Publishing Company, Dordrecht, 1985.

[28] R. H. Möhring and F. J. Radermacher. Introduction to stochastic scheduling problems. In Klaus Neumann and Diethard Pallaschke, editors, *Contributions to Operations Research, Proceedings of the Oberwolfach Conference on Operations Research, 1984*, pages 72–130. Springer-Verlag, Lecture Notes in Economics and Mathematical Systems, vol. 240, 1985.

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

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

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

[32] R. H. Möhring, M. Skutella, and F. Stork. Forcing relations for AND/OR precedence constraints. In *Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00)*, pages 235–236, 2000.

[33] R. H. Möhring, M. Skutella, and F. Stork. Scheduling with AND/OR precedence constraints. Technical Report 689/2000, Technische Universität Berlin, Department of Mathematics, Germany, 2000. A preliminary version of this paper (*Forcing Relations for AND/OR Precedence Constraints*) appeared in [32].

[34] R. H. Möhring and F. Stork. Linear preselective policies for stochastic project scheduling. *Mathematical Methods of Operations Research*, 2000. to appear.

[35] J. H. Patterson, R. Słowiński, F. B. Talbot, and J. Węglarz. An algorithm for a general class of precedence and resource constrained scheduling problems. In R. Słowiński and J. Węglarz, editors, *Advances in Project Scheduling*, pages 3–28. Elsevier, 1989.

[36] PSPLIB. `ftp://ftp.bwl.uni-kiel.de/pub/operations-research/psplib/HTML/data.html`, 2000.

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

[38] F. J. Radermacher. Scheduling of project networks. *Annals of Operations Research*, 4:227–252, 1985.

[39] A. Sprecher. Scheduling resource-constrained projects competitively at modest memory requirements. *Management Science*, 46:710–723, 2000.

[40] F. Stork and M. Uetz. On the representation of resource constraints in project scheduling. Technical Report 693/2000, Technische Universität Berlin, Department of Mathematics, 2000.

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

[42] V. Valls, M. Laguna, P. Lino, A. Pérez, and M. S. Quintanilla. Project scheduling with stochastic activity interruptions. In J. Węglarz, editor, *Handbook on Recent Advances in Project Scheduling*, pages 333–353. Kluwer, Amsterdam, 1998.

[43] J. Węglarz, editor. *Project Scheduling: Recent Models, Algorithms, and Applications*. Kluwer, 1999.

667/2000 *Sándor P. Fekete and Henk Meijer:* On geometric maximum weight cliques

666/2000 *Sándor P. Fekete, Joseph S. B. Mitchell, and Karin Weinbrecht:* On the continuous Weber and k-median problems

664/2000 *Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz:* A note on scheduling problems with irregular starting time costs

661/2000 *Frederik Stork and Marc Uetz:* Resource-constrained project scheduling: from a Lagrangian relaxation to competitive solutions

658/1999 *Olaf Jahn, Rolf H. Möhring, and Andreas S. Schulz:* Optimal routing of traffic flows with length restrictions in networks with congestion

655/1999 *Michel X. Goemans and Martin Skutella:* Cooperative facility location games

654/1999 *Michel X. Goemans, Maurice Queyranne, Andreas S. Schulz, Martin Skutella, and Yaoguang Wang:* Single machine scheduling with release dates

653/1999 *Andreas S. Schulz and Martin Skutella:* Scheduling unrelated machines by randomized rounding

646/1999 *Rolf H. Möhring, Martin Skutella, and Frederik Stork:* Forcing relations for AND/OR precedence constraints

640/1999 *Foto Afrati, Evripidis Bampis, Chandra Chekuri, David Karger, Claire Kenyon, Sanjeev Khanna, Ioannis Milis, Maurice Queyranne, Martin Skutella, Cliff Stein, and Maxim Sviridenko:* Approximation schemes for minimizing average weighted Completion time with release dates

639/1999 *Andreas S. Schulz and Martin Skutella:* The power of α-points in preemptive single machine scheduling

634/1999 *Karsten Weihe, Ulrik Brandes, Annegret Liebers, Matthias Müller–Hannemann, Dorothea Wagner and Thomas Willhalm:* Empirical design of geometric algorithms

633/1999 *Matthias Müller–Hannemann and Karsten Weihe:* On the discrete core of quadrilateral mesh refinement

632/1999 *Matthias Müller–Hannemann:* Shelling hexahedral complexes for mesh generation in CAD

631/1999 *Matthias Müller–Hannemann and Alexander Schwartz:* Implementing weighted b-matching algorithms: insights from a computational study

629/1999 *Martin Skutella:* Convex quadratic programming relaxations for network scheduling problems

628/1999 *Martin Skutella and Gerhard J. Woeginger:* A PTAS for minimizing the total weighted completion time on identical parallel machines

624/99 *Rolf H. Möhring:* Verteilte Verbindungssuche im öffentlichen Personenverkehr: Graphentheoretische Modelle und Algorithmen

627/1998 *Jens Gustedt:* Specifying characteristics of digital filters with FilterPro

620/1998 *Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz:* Resource constrained project scheduling: computing lower bounds by solving minimum cut problems

619/1998 *Rolf H. Möhring, Martin Oellrich, and Andreas S. Schulz:* Efficient algorithms for the minimum-cost embedding of reliable virtual private networks into telecommunication networks