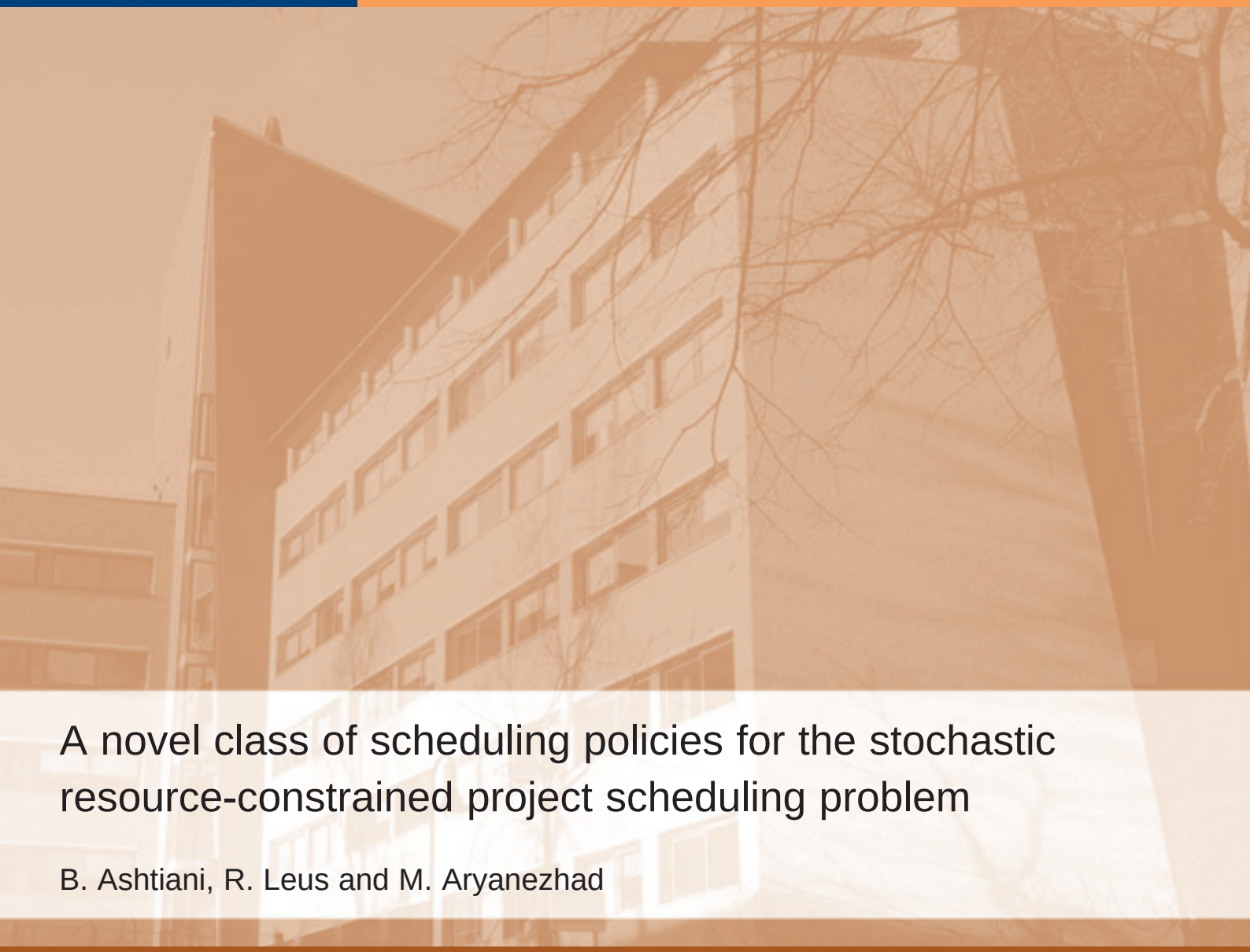
KATHOLIEKE UNIVERSITEIT
LEUVEN

Faculty of Business and Economics



A novel class of scheduling policies for the stochastic resource-constrained project scheduling problem

B. Ashtiani, R. Leus and M. Aryanezhad

DEPARTMENT OF DECISION SCIENCES AND INFORMATION MANAGEMENT (KBI)

KBI 0819

A novel class of scheduling policies for the stochastic resource-constrained project scheduling problem

Behzad Ashtiani
Department of Industrial Engineering,
Iran University of Science and Technology, Tehran, Iran

Roel Leus*
Department of Decision Sciences and Information Management
Katholieke Universiteit Leuven, Belgium

Mir-Bahador Aryanezhad
Department of Industrial Engineering,
Iran University of Science and Technology, Tehran, Iran

— October 2008 —

*Corresponding author. Tel. +32 16 32 69 67. Fax +32 16 32 66 24. E-mail Roel.Leus@econ.kuleuven.be.

A novel class of scheduling policies for the stochastic resource-constrained project scheduling problem

We study the resource-constrained project scheduling problem with stochastic activity durations. We introduce a new class of scheduling policies for this problem, which make a number of a-priori sequencing decisions in a pre-processing phase, while the remaining decisions are made dynamically during project execution. The pre-processing decisions entail the addition of precedence constraints to the scheduling instance, hereby resolving some potential resource conflicts. We compare the performance of this new class with existing scheduling policies for the stochastic resource-constrained project scheduling problem, and we observe that the new class is significantly better when the variability in the activity durations is medium to high.

Keywords: project scheduling, uncertainty, stochastic activity durations, scheduling policies.

---

1 Introduction

Project scheduling is the part of project management that deals with determining when in time to start (and finish) which activities and with the allocation of scarce resources to the project activities. As projects grow in size and complexity, scheduling becomes an increasingly important part of project management. In practice, virtually all project managers are confronted with resource scarceness. In such cases, the Resource-Constrained Project Scheduling Problem (RCPSP) arises. This optimization problem has become popular over the last few decades because of its practical relevance for various industrial and research fields. Numerous procedures have been developed in literature for finding either optimal or heuristic solutions for the RCPSP. For recent surveys, we refer to Demeulemeester and Herroelen [9], Kolisch and Hartmann [26, 27], Kolisch and Padman [28], Neumann et al. [35].

Several exact procedures have been developed to solve the RCPSP, but it is worth noting that it is often quite impractical to solve large-scale instances to guaranteed optimality [5, 8, 37], which can be explained by the fact that the RCPSP is known to be *NP*-Hard [4]. Hence, to solve problems of the size generally experienced in practice, most research efforts focus on the development of heuristic procedures. *Priority rules* and *meta-heuristics* constitute the majority of the developed approaches. Priority rules require little computational effort, which is why they are often employed in literature to solve large problems [24]. Research on priority-rule-based heuristics continues to receive attention in the literature. Xu et al. [49], for instance, combine justification and rollout with a priority rule to solve the RCPSP. Priority rules can also be embedded in other heuristic methods; most meta-heuristic procedures, for example, adopt as representation of solutions either an activity list or a so-called "random-key" vector [26], which is subsequently passed to a schedule generation scheme, which is the basic construct underlying priority rules.

Lambrechts [30] notes that in practice, project parameters are seldom precisely known and usually subject to estimation errors. Uncertainty is the prime cause of these incomplete

and unreliable data. Unfortunately, when uncertainty comes into play, the cited solution procedures for the deterministic RCPSP are no longer valid. This uncertainty can originate from a great number of potential sources [47, 50]; from among many causes, we can cite the deviation of real activity durations from their estimated values, resource unavailability, late material arrivals, changing ready times and due dates, network-structure changes and bad-weather delays. Although the sources of variability in the project environment are manifold, the main scheduling objectives are mostly functions of the activities' starting or ending times, the project makespan being the single most-studied objective, in addition to other ones such as weighted earliness-tardiness and net present value of the project. This justifies a restriction to the study of uncertainty in processing times only, although many different sources may be at the basis of this variability. In this paper, we only focus on makespan minimization.

A recent survey of the various approaches to scheduling under uncertainty is provided by Herroelen and Leus [20, 21]. Three main categories of approaches can be distinguished: proactive, reactive and stochastic. The aim of *proactive scheduling* is to build a *robust* initial schedule, which is as well as possible protected against the influence of potential disruptions. The robustness of a schedule can be defined as the ability to cope with small fluctuations in the input parameters, for instance an increase in the duration of some activities resulting from uncontrolled factors [1]. One common approach to ensure against time variability is buffer insertion. Despite the use of proactive scheduling procedures, disturbances during the project's execution may still cause large deviations from the initial schedule. *Reactive* procedures can then be applied to repair the schedule and to minimize the effect of the disruptions. Reactive scheduling revises or re-optimizes the baseline schedule at hand when an unexpected event occurs. Some sources use the term *predictive-reactive* scheduling, to stress that an initial schedule is constructed that is updated afterwards. Recent work by Van de Vonder et al. [46] looks into the trade-off between reactive and proactive procedures for achieving *quality* and *solution robustness*, which refers to the realized project duration and the deviation between the planned and realized start times of the individual activities, respectively. Chtourou and Haouari [7] develop a two-staged priority-rule-based algorithm in which minimizing the overall makespan is considered in the first stage, after which schedule robustness is maximized.

In methodologies for *stochastic project scheduling*, the scheduling problem is viewed as a multi-stage decision process. These methods define starting times for the activities by using scheduling policies based on a-priori knowledge, possibly based on experience from the past, about processing-time distributions [12, 34, 38]. Scheduling policies gradually build a schedule during the project's implementation and do not construct a complete schedule before project initiation; for this reason, these approaches are sometimes also referred to as *purely reactive* or *on-line* procedures. The aim of this article is to introduce a new class of scheduling policies in which a number of a-priori sequencing decisions are made in a pre-processing phase, and the remaining decisions are made dynamically during project execution. The pre-processing decisions entail the addition of precedence constraints to the scheduling instance, hereby resolving some potential resource conflicts.

The contributions of this article are fourfold: (1) we propose a new class of scheduling policies for project scheduling with stochastic activity durations, extending the existing classes of earliest-start policies and resource-based priority policies; (2) we underline the

value of pre-processing in stochastic scheduling, achieved by making a subset of sequencing decisions at the beginning of the planning horizon and relegating the rest of the scheduling process to future points in time; (3) we develop a local-search procedure to find good policies within the newly proposed class; and (4) for a representative dataset, we obtain computational results that show that the new class of policies achieves high-quality solutions for expected-makespan minimization, outperforming the models available in the literature when the processing-time variability is medium or high.

The remainder of this paper is organized as follows: a number of definitions are provided in Section 2, together with a description of a new class of scheduling policies. Section 3 outlines our search procedure for identifying high-quality members of the new class, the computational results of which are reported in Section 4. A summary and some conclusions are given in Section 5.

2 Definitions and a new class of policies

The basics of the deterministic RCPSP are sketched in Section 2.1, its stochastic counterpart and a discussion of scheduling policies are the subject of Section 2.2. A novel class of policies is introduced in Section 2.3, with further illustrations and discussion in Section 2.4.

2.1 The deterministic RCPSP

An instance of the (deterministic) RCPSP consists of a set of activities $N = \{0, \ldots, n\}$ with known durations $d_i \in \mathbb{N}$ for each activity $i \in N$. Each activity $i \in N$ requires a constant number r_{ik} of each of the renewable resource types k ($k \in K$) during each period of its execution, and each activity has to be processed without interruption. Each resource type $k \in K$ has an availability a_k during each period of the planning horizon.

Precedence constraints are imposed between some pairs of activities, implying that some of the activities can only be started once other activities are finished. These constraints are described by means of an acyclic graph $G(N, A)$, where A is a set of pairs of activities. We assume that A is a (strict) order relation on N, i.e. an irreflexive and transitive relation; henceforth we call A a *precedence relation*. The activities 0 and n are dummy activities representing the start and the end of the project: they have zero duration and resource usage, and are predecessor, respectively successor, of all other activities in N.

A solution to the RCPSP is a *schedule*, which is a vector $\mathbf{s} = (s_0, \ldots, s_n)$ specifying a starting time s_i for each activity $i \in N$. Without loss of generality, we restrict our attention to integer components for the vector $\mathbf{s}$. The time interval $[t-1, t]$ is referred to as (time) period t, for integer t. For period $t \in \mathbb{N}_0$, we define $\mathcal{A}(\mathbf{s}, t) = \{i \in N : s_i \leq (t-1) \wedge (s_i + d_i) \geq t\}$, containing the activities in schedule $\mathbf{s}$ that are *active* during period t. The RCPSP can now conceptually be formulated as follows, where the first constraint set contains the precedence constraints and the second set imposes a limit on the number of resource units that can concurrently be deployed:

$$\text{minimize } s_n$$

subject to

$$\begin{aligned} s_i + d_i &\leq s_j && \forall (i,j) \in A \\ \sum_{i \in \mathcal{A}(\mathbf{s},t)} r_{ik} &\leq a_k && \forall t \in \mathbb{N}_0, \forall k \in K \\ s_i &\geq 0 && \forall i \in N \end{aligned}$$

Numerous heuristics have been developed for producing feasible, high-quality solutions for the RCPSP (see Kolisch and Hartmann [26, 27] for reviews). The fastest heuristics are the *priority rules*, which build a feasible schedule by means of a Schedule Generation Scheme (SGS). An SGS assigns starting time $s_0 = 0$ and then iteratively determines starting times for the other activities. Two types are distinguished in the literature: the parallel SGS and the serial SGS. The *parallel SGS* takes an ordering of the activities as input and starts at time 0, initiates as many activities as possible in the order dictated by the list, and then increments the decision moment and iterates. A *resource-based priority rule* combines the parallel SGS with a specification of the priorities with which activities are selected. Resource-based priority rules produce *non-delay schedules*, which are schedules in which activities cannot start earlier without delaying another activity even if activity preemption were allowed. This set of priority rules has a number of drawbacks. First of all, the set of non-delay schedules need not contain a schedule with minimum makespan [25]. Moreover, changes in activity durations may lead to so-called Graham anomalies [16] such as increasing project duration due to decreasing activity durations.

The *serial SGS* performs activity incrementation instead of time incrementation: a schedule is gradually constructed by adding one activity at a time, until a complete feasible schedule is obtained. Each selected activity is scheduled at the earliest time at which it leads to a resource-feasible partial schedule. An *activity-based priority rule* is an algorithm that outputs a feasible schedule for an RCPSP-instance by specifying the order in which activities are to be considered according to the serial SGS. Kolisch [24] has shown that the serial SGS generates *active schedules*, which are schedules in which none of the activities can be started earlier without delaying another activity. For scheduling problems with regular performance measure, such as makespan minimization, the set of active schedules always contains an optimal solution. We note that the set of active schedules is a superset of the set of non-delay schedules.

The set of such activity-based priority rules is known to contain an optimal algorithm for each RCPSP-instance. Sprecher [40] uses this insight in the development of branch-and-bound procedures for the RCPSP.

2.2 The stochastic RCPSP

In the stochastic resource-constrained project scheduling problem (stochastic RCPSP or SRCPSP), the duration D_i of each activity $i \in N$ is a random variable (rv); the random vector $(D_0, D_1, \ldots, D_n)$ is denoted by $\mathbf{D}$. For the activities $i = 0, n$ we have $Pr[D_i = 0] = 1$; for the remaining activities $i \in N \setminus \{0, n\}$ we assume that $Pr[D_i < 0] = 0$ (where $Pr[e]$ represents the probability of event e).

A solution to the stochastic RCPSP is no longer a schedule but rather a *scheduling policy* (or *strategy*), which generates a schedule as time progresses and knowledge about the processing-time realizations unfolds. Fernandez et al. [13] view a policy as a dynamic decision process: it provides a decision rule that determines which activities have to started at given decision times. These decision times are typically time 0 (the start of the project) and the completion times of activities. A decision at time t can only use information that has become available before or at time t; this requirement is often referred to as the "non-anticipativity constraint". A schedule is thus constructed gradually through time.

A realization of $\mathbf{D}$ is written as $\mathbf{d}$; we will use the terms *realization*, *sample* and *scenario* interchangeably throughout the text. Stork [42], following Igelmund and Radermacher [23], proposes to view a policy as a function: a policy Π is a function $\mathbb{R}_{\geq}^{n+1} \mapsto \mathbb{R}_{\geq}^{n+1}$ that maps given samples $\mathbf{d}$ of activity durations to feasible starting-time vectors (schedules) $\mathbf{s} = \Pi(\mathbf{d}) \in \mathbb{R}^{n+1}$. For a given scenario $\mathbf{d}$ and policy Π, the value $[\Pi(\mathbf{d})]_n$ is the makespan of the resulting schedule (where $[\cdot]_i$ represents the $(i+1)$-th component of the vector between square brackets, since our indexing starts from 0). The project completion time $[\Pi(\mathbf{D})]_n$ is a rv, and the objective for the SRCPSP is to select a policy Π^* that minimizes $E[[\Pi(\mathbf{D})]_n]$ within a specific class $\mathcal{C}$, with $E[\cdot]$ the expectation operator with respect to $\mathbf{D}$.

Various classes of scheduling policies have been proposed for the SRCPSP; for a review of the state of the art, we refer to Stork [42]. Of direct interest to this text are the classes of *resource-based (priority) policies* (*RB-policies*), *activity-based (priority) policies* (*AB-policies*) and *earliest-start policies* (*ES-policies*). The first class $\mathcal{C}^{RB}$ of resource-based policies is a direct extension of the deterministic resource-based priority rules: any $\Pi \in \mathcal{C}^{RB}$ is characterized by an ordering L of the activities[1]. At any decision time, a resource-based policy will consider all unstarted activities in the order of L and start them if this does not violate any precedence nor resource constraint. Since deterministic processing times are merely a special case of stochastic processing times, the disadvantages of resource-based priority rules carry over to this corresponding class of priority policies. In particular, the Graham anomalies illustrate that, when viewed as a function, these policies are neither monotone nor continuous. This fact is referred to by many sources, for instance by Möhring [33], as "unsatisfactory stability behavior" and used as a motivation to eliminate these policies from further study.

We denote the class of AB-policies by $\mathcal{C}^{AB}$. Any $\Pi \in \mathcal{C}^{AB}$ is also represented by a priority list L of the activities and, for a given sample $\mathbf{d}$, computes starting times by starting each activity in the order imposed by L as early as possible, with the side constraint that $[\Pi(\mathbf{d}; L)]_i \leq [\Pi(\mathbf{d}; L)]_j$ if $i \prec_L j$, for $\{i, j\} \subset N$. Elimination of this side constraint would yield a simple resource-based policy that suffers from the Graham anomalies, but the "activity-based" point of view instead of the greedy "resource-based" one does away with this problem. The AB-policies are a logical stochastic counterpart of the activity-based priority rules for the RCPSP, which is why they are sometimes also called "(stochastic) serial SGS" (see e.g. Ballestín [2]). In light of the constraints on the starting times entailed by L, it is required that $i \prec_L j$ for each $(i, j) \in A$ for the corresponding policy to be feasible. Put differently, L defines a linear extension of the input order A.

The class $\mathcal{C}^{ES}$ of ES-policies was introduced by Igelmund and Radermacher [23]. The

[1]In this article, we use both the terms "ordering" and "(order) list" to refer to a complete order on the set N, represented as a permutation $L = (j_0, j_1, \ldots, j_n)$.

simple and intuitive idea is to "break" minimal forbidden sets in order to solve potential resource conflicts. A *forbidden set* $F \subset N$ is a set of activities that are not precedence-related (there is no edge $(i,j) \in A$ for any pair $\{i,j\} \subseteq F$) and that violate a resource constraint if performed concurrently, so $\sum_{i \in F} r_{ik} > a_k$ for at least one $k \in K$. An inclusion-minimal forbidden set is called a *minimal forbidden set* or *mfs*. We denote by $\mathcal{F}(E)$ the set of mfss for precedence relation E. For a binary relation E on N, let $T(E)$ denote its *transitive closure*, defined as the minimal transitive relation on N that contains E. An ES-policy $\Pi \in \mathcal{C}^{ES}$ takes a set of activity pairs $X \subset (N \times N) \setminus A$ as parameter; the idea is to extend partial order A to $A \cup X$ such that $\mathcal{F}(T(A \cup X)) = \varnothing$ or, in other words, so that we can ignore the resource constraints if we respect the extended set of precedence constraints $A \cup X$. The condition for feasibility of the policy is that $G(N, A \cup X)$ still be acyclic. In order to obtain a schedule for a given scenario $\mathbf{d}$ of processing times, an ES-policy Π simply computes earliest starting times with respect to $G(N, A \cup X)$, so by embedding the additional activity pairs into the precedence relation: $[\Pi(\mathbf{d}; X)]_0 = 0$ and

$$[\Pi(\mathbf{d}; X)]_j = \max_{(i,j) \in A \cup X} \{[\Pi(\mathbf{d}; X)]_i + d_i\} \qquad \forall j \in N \setminus \{0\}$$

One direct computational benefit of priority policies over ES-policies is that algorithmic procedures do not require the examination of all minimal forbidden sets, the number of which may be exponential in the number of activities. Stork [42] compares the optimal expected project makespan for the different classes. If we define ρ^{RB}, ρ^{AB} and ρ^{ES} to be the optimal objective-function values when optimizing over the classes $\mathcal{C}^{RB}$, $\mathcal{C}^{AB}$ and $\mathcal{C}^{ES}$, respectively, then we have $\rho^{ES} = \rho^{AB} \leq \rho^{RB}$ in the deterministic case. In the stochastic case, however, the behavior is quite different: Stork provides a number of examples that show that each pair of classes from $\mathcal{C}^{RB}$, $\mathcal{C}^{AB}$ and $\mathcal{C}^{ES}$ is incomparable, meaning that in some instances $\rho^{RB} < \rho^{AB}$ and in other cases $\rho^{RB} > \rho^{AB}$, and similarly for the other comparisons.

In the remainder of this text, we describe a new set of policies that encompasses $\mathcal{C}^{RB}$ and $\mathcal{C}^{ES}$, and we disregard $\mathcal{C}^{AB}$. In spite of the aforementioned reasons for some authors not to investigate $\mathcal{C}^{RB}$, we will consider RB-policies in our search for a policy with low expected makespan. One main objection against RB-policies, namely that they might "miss" the optimum in deterministic scheduling, becomes invalid by considering the set of pre-processor policies (formally defined in Section 2.3), since this new class of policies is a superset of $\mathcal{C}^{ES}$. Additionally, we are not very much concerned with the stability properties of the policy at hand – actually, we conjecture that such (lack of) stability hardly, if ever, constitutes an issue to a practical decision maker when the expected makespan is appropriately low. A more detailed motivation for the choice for RB-policies follows in Section 2.4. One different element that might create concerns to the decision maker is the *variability* of the outcomes but, to the best of our knowledge, variability has by itself to date never been studied in the context of stochastic resource-constrained project scheduling except for Ballestín and Leus [3], who observe a trade-off between expected makespan and makespan variability. For the record, we should point out that makespan variance has been studied in the absence of resource constraints, notably by Elmaghraby et al. [11], who distinguish both mean and variance of the project duration as the two prime performance measures of concern, by Gutierrez and Paul [17], who examine the impact of variability in activity durations on mean project duration, and by Cho and Yum [6], whose focus is more on the sensitivity of makespan variability.

2.3 Pre-processor policies

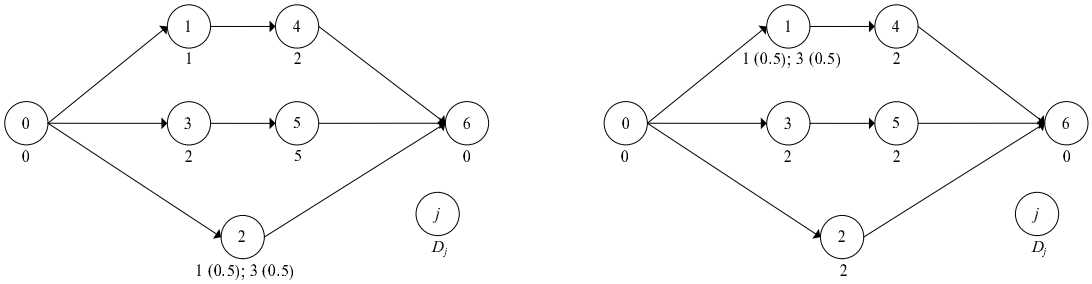
Wu et al. [48] argue that "when and how to make which decisions" represents the most crucial aspect of planning and scheduling in situations where unforeseen disturbances occur. They identify a critical subset of scheduling decisions that, to a large extent, dictate global schedule performance and that are made at the beginning of the planning horizon, while the remainder of the scheduling decisions is relegated to future points in time. In the disjunctive-graph representation of the classical job-shop scheduling problem, these pre-processing decisions entail the orientation of a number of disjunctive arcs; the resulting partially oriented disjunctive precedence graph is then used as a basis for the remaining scheduling decisions, which are made progressively throughout the planning period by means of a dynamic dispatching rule.

The intuition underlying this *Pre-process First Schedule Later* (PFSL) scheme can be translated into the setting of the SRCPSP as follows. It is well known, see for instance Leus and Herroelen [31], that ES-policies constitute an extension of the disjunctive-graph view of the job shop to a project scheduling environment: each additional precedence constraint constitutes an extra sequencing decision that is imposed before the project starts. The class of ES-policies only contains policies that resolve *all* mfss, leaving only a trivial scheduling problem without further potential resource conflicts. We propose the new class $\mathcal{C}^{PP}$ of *pre-processor policies* (*PP-policies*), which combine some of the unconditional sequencing decisions by members of $\mathcal{C}^{ES}$ with the real-time dispatching features of the elements in $\mathcal{C}^{RB}$. A PP-policy $\Pi \in \mathcal{C}^{PP}$ is defined by a set of activity pairs $X \subset N \times N$ *together with* an activity list L; Π is feasible if and only if $G(N, A \cup X)$ is acyclic. The policy Π may resolve some but necessarily all resource conflicts before the execution of the project: $0 \leq |\mathcal{F}(T(A \cup X))| \leq |\mathcal{F}(A)|$. All remaining resource conflicts are dynamically resolved during project execution by an RB-policy defined by L for precedence graph $G(N, A \cup X)$.

2.4 Illustrations and discussion

We present a few small example projects to illustrate the newly introduced class of policies. The project network depicted in Figure 1(a) (in activity-on-the-node format) contains five non-dummy activities with activity durations as indicated (the network shown is actually the transitive reduction of the graph $G(N, A)$, or in other words: the indirect arcs are not included). The resource usage of the individual activities is unimportant to this illustration, we only specify the set of mfss as $\mathcal{F}(A) = \{\{2, 4, 5\}\}$. The optimal ES-policy for this instance has expected makespan equal to 7 (average of 7 and 7) and adds the activity pair $(4, 2)$ to the precedence network; multiple priority lists lead to a best expected makespan of 7.5 (average of 7 and 8) among the class of RB-policies, one such list is $(0, 1, 2, 3, 4, 5, 6)$.

Figure 1(b) presents a very similar scheduling instance, with the same number of activities and precedence network, but the durations and resource constraints are slightly altered. We again consider only one mfs, which is now $\{4, 5\}$. We now have optimal objective-function value 6 for the best ES-policies (which add either $(4, 5)$ or $(5, 4)$ to the network), versus 5.5 for the best RB-policy (corresponding with the list $(0, 1, 2, 3, 4, 5, 6)$). The foregoing optimal policies were found by means of full enumeration. We observe that none of the two classes $\mathcal{C}^{RB}$ and $\mathcal{C}^{ES}$ dominates the other, and since $(\mathcal{C}^{RB} \cup \mathcal{C}^{ES}) \subset \mathcal{C}^{PP}$, the new class $\mathcal{C}^{PP}$ strictly



(a) An ES-policy is the best

(b) An RB-policy is the best

Figure 1: Two example projects, one for which an ES-policy is the best and one for which an RB-policy is the best. When the duration D_i of an activity i can take on more than one value then all possible values are listed, followed by their probability (between brackets).

dominates each of the other two.

Based on the discussion in the foregoing sections on potential disadvantages of RB-policies, one might be tempted to combine $\mathcal{C}^{ES}$ with $\mathcal{C}^{AB}$ rather than with $\mathcal{C}^{RB}$, so to resolve part of the resource conflicts via sequencing decisions and then apply an AB-policy to the extended precedence network. We illustrate why this is not preferable by means of the example project represented in Figure 2. The activity durations follow a continuous uniform distribution with variance equal to 3 and $E[D_i]$ equal to $4, 3, 3, 5$ and 6 for $i = 1$ to 5, and $|K| = 1$ with $a_1 = 2$ and $r_i \equiv r_{i1}$. The best ES-policy is strictly better than the best AB and RB-policies for this instance; this optimal policy corresponds with the inclusion of the single additional arc $(3, 2)$ in the network. The objective function was evaluated using simulation.

If we were to implement a PP-policy with arc selection $\{(3, 2)\}$ by applying an AB-policy to the extended network then we would start the two activities in $\{1, 3\}$ together at time 0 if we wished to mimic the corresponding ES-policy. Subsequently, dependent on the realization of the durations D_1 and D_3, new activities become eligible; suppose that $D_1 < D_3$. In order for our PP-policy to adopt the behavior of the ES-policy, activity 4 should be processed immediately after the end of activity 1; this means that the activity list for the PP-policy needs to be of the form $(0, 1, 3, 4, -, -, 6)$ or $(0, 3, 1, 4, -, -, 6)$. If this is the case, however,

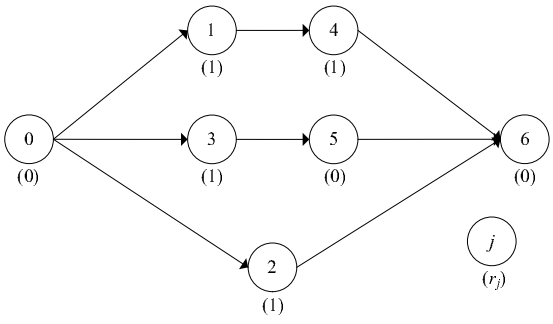


Figure 2: An example project showing why we do not use AB-policies.

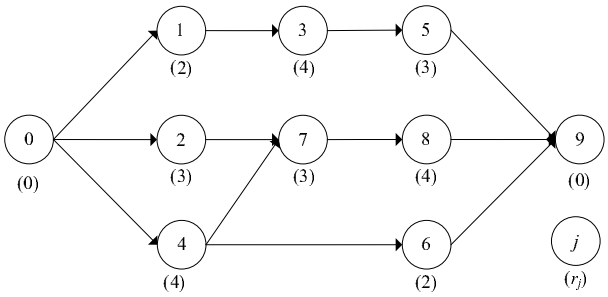


Figure 3: A project instance that demonstrates that PP-policies can be strictly better than both ES-policies and RB-policies.

then we encounter a problem in the scenarios with $D_1 > D_3$: ES-policies will start both activity 2 as well as activity 5 immediately at the end of activity 3, while this is impossible for a PP-policy with a list of the described format. Consequently, the extra constraints on the starting times of the activities that are imposed by the AB-policies would preclude the corresponding class of PP-policies from being a proper generalization of the class of ES-policies.

Consider the instance depicted in Figure 3, which contains eight non-dummy activities with expected durations equal to $2, 7, 3, 4, 8, 6, 4$ and 2, respectively. Each of the activities has a stochastic duration with continuous uniform distribution with variance equal to 3. The resource usage for each activity is shown below the corresponding node; we are dealing with a single resource type with an availability of eight units. The set of mfss is $\mathcal{F}(A) = \{\{1,2,4\},\{2,3,4\},\{2,3,6\},\{2,4,5\},\{3,6,7\},\{3,6,8\},\{5,6,8\}\}$. The optimal ES-policy for this instance corresponds with an expected makespan of 18.69 and adds the activity pairs $(1,2)$, $(4,3)$, $(3,6)$, $(4,5)$ and $(5,8)$ to the precedence network; multiple priority lists lead to a best expected makespan of 17.73 among the class of RB-policies, one such list is $(0,1,2,3,4,5,6,7,8,9)$. Both of these solutions (which are the best in the classes $\mathcal{C}^{ES}$ and $\mathcal{C}^{RB}$, respectively), are outperformed by the PP-policy corresponding with the addition of the single activity pair $(1,6)$ and using the list $(0,1,4,2,3,6,5,7,8,9)$, which leads to an expected makespan of 17.54. Note also that the edge $(1,6)$ does not resolve any mfs! This additional precedence constraint does entail a sequencing decision that apparently creates convenient combinatorial opportunities for filling the resource profile in the majority of the scenarios.

3 A two-phase local-search procedure for pre-processor policies

The population-based adaptive search procedures known as genetic algorithms (GAs), introduced by Holland [22], serve as a heuristic meta-strategy to solve hard optimization problems. For a detailed discussion on GAs we refer to Goldberg [14]. A GA starts with the construction of an initial population (often called "first generation") and computes the next generations by applying crossover, mutation and selection operators. The initial population is randomly divided into pairs (parents); the crossover operator then produces two new individuals (chil-

Algorithm 1 Overall structure

```
ElectList = ListGA()
for i = 1 to NoList do
  L = ElectList(i)
  CurSolArc = initial population of candidate activity pairs for list L
  ArcGA(L,CurSolArc)
end for
Return the best solution found
```

dren) per pair. Subsequently, the mutation operator is applied to the resulting offspring. The newly generated individuals are added to the current population after determination of their fitness value. Lastly, the next generation is created by invoking the selection operator, which determines which individuals are carried over to the next generation and which are eliminated. The algorithm terminates when a termination criterion is met; this criterion is generally a pre-defined number of generations or time limit. Below, we discuss GA as a heuristic search procedure for order lists (Section 3.2) and for activity pairs (Section 3.3); we first describe the overall structure of our search procedure for SRCPSP-solutions in Section 3.1.

3.1 Global structure of the algorithm

As outlined in Section 2.3, a PP-policy is fully determined by an order list and a set of activity pairs. In this section, we devise a two-phase GA; in the first phase, good order lists are found by means of a GA without considering extra pairs. Subsequently, in phase two, we search for activity pairs that can improve the quality of the order lists. The overall structure of our GA-implementation is depicted in Algorithm 1. We first run the function ListGA to obtain a set of $NoList$ good activity lists ElectList. Subsequently, a set of high-quality arc selections is found by means of function ArcGA applied to each order list in ElectList. Section 3.2 provides further details on the function ListGA; the function ArcGA is extensively treated in Section 3.3.

3.2 Phase 1: a local-search procedure for order lists

Local-search procedures in which solutions are represented by lists are very common. This part of our algorithm is configured based on a number of well-known sources from the scheduling literature. A general overview of the procedure is shown in Algorithm 2. The key element of the procedure is the ListCrossMut-function, which applies crossover and mutation to the current set of solutions stored in CurSolList. In order to improve the quality of the solutions, we incorporate the technique of justification. A more detailed description of the main parameters and operators follows.

Individuals and fitness. In our search procedure, each individual is a precedence-feasible activity list. An RB-policy transforms an activity list L into a schedule $\Pi(\mathbf{D}, L)$ by scheduling as many eligible activities as possible (within the resource constraints) at each decision point, dependent on the realizations of the vector $\mathbf{D}$. The associated

Algorithm 2 ListGA()

```
CurSolList = initial population of lists
ElectList = the NoList best elements of CurSolList
while TerminationCriterion not met do
  NewSolList = ListCrossMut(CurSolList)
  for i = 1 to NoPList do
    L = NewSolList(i)
    s = Π(E[D]; L)
    s = Justification(s)
    L = ScheduleToList(s)
    Compute the fitness value of order list L
    if L is better than the worst solution L' in ElectList then
      ElectList = (ElectList \ L') ∪ L
    end if
  end for
  CurSolList = Selection(CurSolList,NewSolList)
end while
Return ElectList
```

expected makespan $E[[\Pi(\mathbf{D}, L)]_n]$ is the fitness of the individual L, which is estimated via simulation. Ballestín [2] shows that using fewer scenarios in the determination of the fitness of each activity list leads to better solutions at the end of the procedure because it favors the evaluation of more policies. Consequently, when computational effort matters, we opt for a rather low number n_{sim} of replications for evaluation during the search, leading to lower accuracy but a higher number of scanned solutions, which usually results in a better final outcome. For the evaluation of the quality of the final policy that is output by the algorithm, on the other hand, accuracy is obviously vital, and so we use 1000 replications.

Initial population. The initial population is generated by employing regret-based biased random sampling (RBRS) [10]. Starting with an empty order list, each next activity at each decision point is randomly selected from the set of eligible activities, which are the unselected activities, all predecessors of which have already been included in the list. The probability π_j of selection of an activity j out of the eligible set E is determined by means of priority values v, as follows:

$$\pi_j = \frac{(\rho_j + 1)^\alpha}{\sum_{k \in E}(\rho_k + 1)^\alpha},$$

with $\rho_j = \max_{k \in E} v(k) - v(j)$. In our implementation, we fix $\alpha = 1$ and v are the latest finish times.

Crossover. We apply the two-point crossover that was developed by Hartmann [18]. This operator combines a pair of lists into two new lists. First, two individuals are selected as parents (mother and father) and two random integers r_1 and r_2 are drawn, with $1 \leq r_1 < r_2 \leq n$. The daughter (son) is constructed by copying the first r_1 positions from

the mother (father), the positions between r_1 and r_2 are taken from the father (mother), and finally, the remaining positions are again drawn from the mother (father).

Mutation. For a given activity permutation $L^* = (l_0, l_1, \ldots, l_n)$, a standard mutation operator [18] is applied that changes the activity order, as follows. For all positions $i = 1, \ldots, n-1$, the activities l_i and l_{i+1} are exchanged with a predefined probability p^L_{mut}.

Justification. After crossover and mutation, we apply *double justification* to each list. A schedule **s** is first built by applying the parallel SGS with the expected duration of the activities to the list. A double justification consists of shifting activities to the right as far as possible in non-increasing order of their finish times without altering the start of activity n, and then re-shifting them to the left. The principles of justification were described by Li and Willis [32] and Özdamar and Ulusoy [36]. Valls et al. [44] show that justification is an effective technique that can be incorporated in diverse algorithms to enhance the quality of the results without requiring substantially more computation time. In addition, since the set of active schedules dominates the non-delay ones, the solution space becomes larger during the justification, which produces active schedules. Subsequently, the justified schedule is re-converted into an order list by means of the function ScheduleToList by ordering activities by their starting times.

Selection. Hartmann [18] shows that simple ranking outperforms other selection operators (e.g. proportional selection, 2-tournament selection, ...); it is also used by Valls et al. [45] as a good selection operator. Simple ranking keeps the $NoPList$ best individuals and removes the remaining ones from the population (ties are broken arbitrarily).

3.3 Phase 2: a local-search procedure for activity pairs

Contrary to the search procedure for activity lists, the local search for activity pairs is one of the novelties in this paper. The problem we solve is that of finding a set of activity pairs that improves the overall quality of a given list. The function ArcGA (an overview of which is provided in Algorithm 3) is applied to each of the $NoList$ solutions in ElectList, which is the output of ListGA. In each iteration of ArcGA, a new set of extra pairs is produced via crossover and mutation in the function ArcCrossMut. In each generation, the selected list is combined with each of the NoP arc selections, yielding a complete PP-policy, which is evaluated via simulation. Lastly, a selection operator is utilized (the Selection-function) to choose NoP solutions from (CurSolArc) $\cup$ (NewSolArc) to form the next generation. We describe some elements of the procedure in more detail below.

Individuals. In this part of the search procedure, an individual $X = \{x_1, \ldots, x_m\}$ is an (unordered) set of ordered activity pairs $x_i = (j, k)$, with for $i = 1, \ldots, m : x_i \notin A$. This set is said to be feasible if and only if $G(N, A \cup X)$ is acyclic.

Initial population. For the initial population, we need to generate subsets of the set $\bar{A} = \{(i, j) \in N \times N | (i, j) \notin A \wedge (j, i) \notin A\}$. The cardinality of $\bar{A}$ is quadratic in n; for the project depicted in Figure 3 with just nine non-dummy activities, for instance, $\bar{A}$

Algorithm 3 ArcGA(L,CurSolArc)

```
CurSolArc = initial population of sets of activity pairs
while TerminationCriterion not met do
  NewSolArc = ArcCrossMut(CurSolArc)
  for i = 1 to NoP do
    X = NewSolArc(i)
    Compute the fitness value of PP-policy parameters (L, X)
    if (L, X) is better than the currently best found solution then
      remember (L, X) as new best solution
    end if
  end for
  CurSolArc = Selection(CurSolArc,NewSolArc)
end while
```

contains 40 activity pairs. Each population member contains either one or two activity pairs, each possibility being chosen with 50% probability.

We propose four approaches to the creation of the initial population. The first is the *Sampling Arc Method* (SAM), in which we apply RBRS with $\alpha = 1$ for obtaining the probabilities of selection of each arc, in which the priority values v are the makespans obtained for the expected durations with the arc selection consisting of only the arc in question.

In a second procedure, called *Time-Based Method* (TBM), we apply an RB-policy to the scheduling instance with expected durations, where at each decision point t we encounter a set $N_t \subset N$ of activities that are either eligible (all predecessors have finished) or in process. All activity pairs (i, j) that can improve the makespan, with $\{i, j\} \subseteq N_t$ for any decision point t, constitute a set C of candidate arcs (in SAM, C is equal to $\bar{A}$); the initial population is then constructed from C via RBRS. The potential for makespan improvement is tested via the condition $[\Pi(E[\mathbf{D}], L, \{(i, j)\})]_n < [\Pi(E[\mathbf{D}], L, \varnothing)]_n$.

In the third procedure, called *Sampling Forbidden Sets Method* (SFSM), we first determine only a subset $\mathcal{F}_P(A) \subseteq \mathcal{F}(A)$ of the mfss that will be considered for being broken. As for TBM, we apply an RB-policy to the scheduling instance with expected durations, and at each decision point t we encounter a set $E_t \subset N$ of activities that are eligible. All mfss $F \subseteq E_t$ are included in $\mathcal{F}_P(A)$, for all decision points t. In this way, we hope to distinguish only the mfss with a significant probability of occurrence during project execution. Subsequently, we need to compose a set C of candidate arcs. To this aim, we also limit the resolution possibilities for each mfs: for each pair $\{i, j\} \subseteq F$ with $F \in \mathcal{F}_P(A)$, the edge (i, j) is added to C if $[\Pi(E[\mathbf{D}], L, \{(i, j)\})]_n < [\Pi(E[\mathbf{D}], L, \varnothing)]_n$ (with L the list under consideration).

A fourth and last approach to the generation of the initial population is the *Forbidden pairs method* (FPM), which combines ideas of SFSM and TBM and which considers left-over (i.e., unused) capacity of resources. At each decision point t, we have a set E_t of eligible activities and a set W_t of activities in process; the left-over capacity of resource type k is defined as $a'_k = a_k - \sum_{i \in W_t} r_{ik}$. We determine a set of forbidden pairs

$FP = \{(i,j) \in E_t, \exists k \in K : r_{ik} + r_{jk} > a'_k\}$; each of these pairs is added to set C if it can improve the quality of makespan. Subsequently, the pairs $\{(l,i)|l \in W_t, i \in FP\}$ are also added if they can decrease the makespan value.

In order to give the reader an idea of the order of magnitude of the size of the sets of candidate arcs corresponding with each of the foregoing four approaches, we have averaged the size for the first ten instances of the J120-dataset (further details on our computations are provided in Section 4). For SAM, TBM, SFSM and FPM, the average cardinality of C is 12954.4, 212.5, 21 and 128, respectively.

Crossover. The crossover operators for lists cannot be applied here; we propose a *uniform crossover*, as follows. After randomly subdividing the current population into pairs of parents (father and mother) (X_F, X_M), each $x_i \in X_F$ is assigned to the son X_S with a probability of p_{cross}, otherwise it is added to the daughter X_D. All elements $x_i \in X_M$ are analogously included into either X_D or X_S.

Mutation. The mutation operator modifies each individual, which is a set of activity pairs. Let C represent the set of candidate arcs for the initial population; for a given solution X we define its complement $X' = C \setminus X$. Each solution X is mutated with probability p^A_{mut}. If mutation occurs then, with probability p_x, the mutation operator removes one randomly selected pair. Alternatively, with probability $1 - p_x$, a random pair $x \in X'$ is added to X.

Selection. The selection operator is the same as in Phase 1: the solutions for a given list are ranked based on their objective-function estimate, and the first NoP solutions are retained as a new generation.

4 Computational results

In this section, we present the results of our computational experiments with the search for good PP-policies, as set out in the previous section. All experiments were performed on a personal computer with 2,130 MHz clock speed and 1.99 GB RAM. The coding was performed in C using the Microsoft Visual Studio 2005 programming environment under the Windows-XP operating system. The algorithms have been evaluated on the problem set J120, which is a standard dataset for project scheduling, containing 600 RCPSP-instances with 120 non-dummy activities each; this dataset has been generated using the ProGen data generator [29].

We follow Stork [42] and Ballestín and Leus [3], two of the few references within stochastic project scheduling that report computational results on reasonably sized datasets, in the choice of the probability distributions: we examine Uniform, Exponential and Beta-distributed rvs. The deterministic processing times $\mathbf{d}^*$ that appear in J120 are the expected values for the durations, and we work with five distributions: two continuous Uniform distributions with support $[d^*_i \pm \sqrt{d^*_i}]$ and $[0; 2d^*_i]$; one Exponential distribution with expectation d^*_i; and two Beta distributions with variance $d^*_i/3$ and $d^{*2}_i/3$, both with support $[d^*_i/2; 2d^*_i]$. In the following paragraphs, we will refer to these five distributions as U1, U2, Exp, B1 and B2, respectively. The variance of these distributions is, in the same order, $d^*_i/3$, $d^{*2}_i/3$,

Table 1: Average percentage distance of the expected makespan from the critical-path length of the project.

Procedure	# schedules	Distribution				
		U1	U2	Exp	B1	B2
SAM	1×10^4	48.88	58.23	75.64	48.93	58.79
	5×10^4	47.27	57.91	74.95	47.37	58.12
SFSM	1×10^4	48.92	58.14	75.53	48.81	58.64
	5×10^4	47.39	57.84	74.87	47.29	57.91
TBM	1×10^4	48.72	58.05	75.08	48.74	57.99
	5×10^4	47.07	57.70	73.97	47.04	57.60
FPM	1×10^4	48.86	58.06	75.11	48.79	57.98
	5×10^4	47.11	57.71	74.00	47.13	57.66

d^{*2}, $d_i^*/3$ and $d_i^{*2}/3$. This means that U1 and B1 display relatively little variability in the activities' processing times, U2 and B2 have medium variability and Exp corresponds with large variability. For completeness, we state the parameters (α, β) of the Beta distributions: $\alpha = (d_i^*/2) - (1/3)$ and $1/6$ for B1 and B2, respectively, while β is equal to 2α in both cases.

We evaluate each solution (L, X) by means of simulation. Ballestín [2] shows that using fewer scenarios to approximate statistics of each solution during the algorithm enables the evaluation of more solutions, which can lead to better solutions at the end of algorithm. In the literature, scenarios are generally calculated by simple Monte-Carlo sampling. Saliby [39] observes that in Monte-Carlo applications, using simple random sampling, sample moments will vary at random and may yield an imprecise description of the known input distribution, which will increase the variance of the simulation estimates. This shortcoming becomes more severe when using fewer scenarios, which is exactly our choice; for this reason, we resort to *descriptive sampling*. This technique was introduced by Saliby [39] as a variance-reduction technique and works with a random permutation of quantiles of the distribution at hand.

4.1 Configuration of the local search

We evaluate the quality of an algorithm by the average percentage distance (using $1,000$ replications) of $E[[\Pi(\mathbf{D})]_n]$ from the critical-path length with deterministic durations $\mathbf{d}^*$. In order to eliminate the impact of different computer infrastructure when comparing multiple algorithms, computational effort is measured by the the number of generated schedules, which is an accepted method in literature [19]. Ballestín and Leus [3] impose two different upper bounds on the number of schedules examined, namely $5,000$ and $25,000$, but one scheduling pass of an AB-policy is counted as 0.5, which leads to a total of $10,000$ and $50,000$ "actual" scheduling runs; we will adopt these two latter limits in our tests.

Table 1 reports on the comparison of the four approaches (SAM, SFSM, TBM and FPM) to the creation of a set of candidate arcs. In our algorithm, computational effort needs to be divided between the search for lists and the search for extra pairs. We assign 20% of the generated schedules to the pursuit of good lists and the remainder to find suited extra

Table 2: Impact of the number of replications for evaluation of each policy.

Procedure	# schedules	Distribution				
		U1	U2	Exp	B1	B2
5 repl	1×10^4	49.59	58.29	75.56	49.76	58.21
	5×10^4	48.05	57.99	74.60	48.54	57.83
100 repl	1×10^4	50.07	58.65	75.87	50.76	58.24
	5×10^4	48.66	58.03	75.02	48.69	57.95
500 repl	1×10^4	53.61	59.75	76.44	54.08	59.12
	5×10^4	49.73	58.59	75.72	49.73	58.21
1,000 repl	1×10^4	54.11	61.68	79.84	54.57	61.28
	5×10^4	51.45	59.13	76.01	52.83	58.92

pairs. In order to introduce diversity into the search procedure, we have worked with the five best lists rather than with just one (we set $NoList = 5$); in this way, 16% of the generated schedules is devoted to the improvement of each list (by means of additional precedence constraints).

Our local search for high-quality lists adopts one of the best performing configurations encountered in literature [18]. For $50,000$ schedules, the number of individuals $NoPList$ and the number of generations in ListGA are set to 40 and 25, respectively; for $10,000$ schedules, these numbers are fixed to 20 and 10. The probability of mutation p^L_{mut} is set to 0.05 in both cases. We observe that SFSM and SAM are dominated by FPM and TBM, although the differences are minor. We find that TBM leads to better results while consuming more time for evaluating all activity pairs at each decision point, while FPM yields decent results in less time by evaluating a set of activity pairs that could violate left-over capacity at each decision point. In the remainder of the experiments, we implement FPM. The probabilities p_{cross} and p_x are chosen as 0.5.

During the search procedure, we opt for 10 as the number of replications (n_{sim}) for evaluation of each policy, rather than 100 or more. In this way, the algorithm can generate and evaluate more solutions for the same computational effort. We include Table 2, where the trade-off between the number of replications and the quality of the results is studied. The table shows that it is preferable to use only few replications, an observation that is in line with Ballestín and Leus [3]. This choice may lead to low accuracy but corresponds with a higher number of generated and tested policies, yielding a better final outcome.

The relationship between population size (NoP) and number of generations ("GEN") for ArcGA is depicted in Table 3, which shows that the best result is obtained by the values 10 and 80, respectively, in case of 5×10^4 schedules. For 1×10^4 schedules, there are only slight differences between the examined options; the best algorithm corresponds with four generations and a population size of 40. Finally, we have compared different choices for the mutation probability in Phase 2 (p^A_{mut}). Table 4 summarizes our findings; we conclude that the value 0.05 is the best.

Table 3: Impact of the population size.

NoP	GEN	# schedules	Distribution				
			U1	U2	Exp	B1	B2
10	80	5×10^4	47.11	57.71	74.00	47.13	57.66
20	40	5×10^4	47.34	57.89	74.42	47.34	57.91
40	20	5×10^4	47.42	57.86	74.68	47.51	58.01
80	10	5×10^4	47.36	57.82	74.59	47.43	57.97
4	40	1×10^4	48.86	58.06	75.11	48.79	57.98
10	16	1×10^4	49.18	58.41	75.41	49.01	58.15
20	8	1×10^4	49.08	58.36	75.28	48.94	58.05

Table 4: Impact of the mutation probability.

p^A_{mut}	# schedules	Distribution				
		U1	U2	Exp	B1	B2
0.01	1×10^4	49.02	58.41	75.63	48.93	58.26
	5×10^4	47.34	58.17	74.66	47.31	58.06
0.05	1×10^4	48.86	58.06	75.11	48.79	57.98
	5×10^4	47.11	57.71	74.00	47.13	57.66
0.1	1×10^4	49.01	58.91	75.83	49.61	58.69
	5×10^4	47.85	57.91	74.91	47.78	57.86

4.2 Comparison with other algorithms

In this section, we compare the PP-policies produced by our two-phase GA (subsequently referred to as "PPGA") with the state-of-the-art algorithms for the SRCPSP that are available in the literature. First of all, we consider the genetic algorithm of Ballestín [2] and the GRASP algorithm described by Ballestín and Leus [3]; both of these algorithms scan the set of AB-policies, and are named "ABGA" and "ABGR" in what follows. These sources work with the same dataset, schedule limits and distributions. Table 5 depicts the results; PPGA outperforms ABGA in all cases. The benefit of PPGA over ABGR is slightly more mitigated, in that PPGA dominates ABGR in the medium and high-variability cases (U2, B2 and Exp), while for the low-variability distributions U1 and B1, ABGR is better than PPGA, but only by a very small margin.

Secondly, we consider the tabu search (TS) and simulated annealing (SA) procedures of Tsai and Gemmill [43], who evaluated their algorithms on the Patterson dataset [37]. The activity processing times are Beta-distributed rvs, the parameters of which are set using three time estimates as suggested by the PERT-model. Tsai and Gemmill report the deviation from an approximate lower bound (LB). Our results for the same dataset and distribution are in Table 6. PPGA outperforms SA1, SA2, TS1 and TS2 in quality (SA2 and TS2 differ from SA1 and TS1 only in the parameters settings). Ballestín and Leus [3] also test their GRASP procedure for the same instances, they report 2.01% and 1.96% for two variants of the algorithm. The difference between PPGA and ABGR is small, which might be due to

Table 5: Comparison between ABGA, ABGR and PPGA.

Procedure	# schedules	Distribution				
		U1	U2	Exp	B1	B2
ABGA	1×10^4	51.49	78.65	120.22	—	—
	5×10^4	49.63	75.38	116.83	—	—
ABGR	1×10^4	46.84	72.58	114.42	47.17	75.97
	5×10^4	45.21	70.95	112.37	45.60	74.17
PPGA	1×10^4	48.86	58.06	74.96	48.79	57.98
	5×10^4	47.11	57.71	74.00	47.14	57.66

Table 6: Comparison between PPGA and the algorithms of Tsai and Gemmill [43].

Algorithm	SA1	SA2	TS1	TS2	GAPP	
					1×10^4	5×10^4
Above approximate LB	3.40%	2.27%	3.71%	2.54%	2.08%	2.03%

fact that the solutions in both procedures are near optimal.

Golenko-Ginzburg and Gonik [15] test their procedures on only one instance, which has 36 activities and a single renewable resource type. The authors introduce two approaches, one based on an exact procedure to solve consecutive multi-dimensional knapsack problems (Heuristic 1) and one that resorts to heuristic solution of those knapsack problems (Heuristic 2). Table 7 contains their and our results for three duration distributions. This instance is also studied by Stork [41] for the uniform distribution; he obtains 434 as expected makespan (which is achieved by an optimal AB-policy).

4.3 Comparison with RB and ES-policies

As a final means to assess the potential of the newly introduced class of pre-processor policies, we have used RanGen1[2] to generate a dataset with ten scheduling instances with $n = 10$

[2]Available at http://www.projectmanagement.ugent.be/rangen.php.

Table 7: Expected makespan for the instance from Golenko-Ginzburg and Gonik [15]

Algorithm	Distribution		
	Beta	Uniform	Normal
Heuristic 1	433.38	448.49	448.85
Heuristic 2	447.98	461.35	461.58
ABGR (1×10^4)	408.75	427.64	422.04
ABGR (5×10^4)	403.16	424.28	415.40
PPGA (1×10^4)	408.12	427.32	425.66
PPGA (5×10^4)	403.02	423.88	418.79

Table 8: Comparison between heuristic PP-policies and optimal ES and RB-policies.

Procedure	Distribution				
	U1	U2	Exp	B1	B2
ES-policy	29.64	37.96	44.77	28.76	35.05
RB-policy	29.74	37.13	43.44	28.10	32.60
PPGA (5×10^4)	28.51	34.81	42.94	27.51	32.28

(eight non-dummy activities) and $K = 3$ for each combination of the following parameter values: order strength $OS = 0.3, 0.5, 0.75$; resource factor $RF = 0.45, 0.9$; and resource constrainedness $RC = 0.3, 0.6$; this results in a total of 120 instances. We have compared the heuristic PP-policy output by our PPGA-procedure with an *optimal* ES and RB-policy for each instance in the dataset (these optimal policies were found using full enumeration). Table 8 presents the percentage distance of each policy from the deterministic critical-path length. It turns out that the heuristic PP-policies are better than the optimal policies from the two smaller classes, and this for all five the distributions.

5 Summary and conclusions

In this article, we have proposed the new class of *pre-processor* policies for the stochastic resource-constrained project scheduling problem (SRCPSP). These new policies combine elements of *resource-based* and *earliest-start* policies, in that they make a number of initial sequencing decisions in a pre-processing phase in the same way as the earliest-start policies, but contrary to these latter policies, *not all* potential resource conflicts are necessarily resolved. The remaining decisions are made dynamically during the project's execution by adhering to a resource-based policy.

A two-phase local-search procedure is developed to produce high-quality pre-processor policies for SRCPSP-instances. The first phase of this procedure is devoted to finding good priority lists; each of these lists is subsequently paired with a suited set of ex-ante imposed sequencing choices. Our computational experiments indicate that good members of the new class of policies (without guarantee of optimality) outperform *optimal* resource-based and earliest-start policies for small instances. For larger scheduling instances, our solutions are significantly better than those produced by the algorithms available in the literature, for the case of medium to high variability. When the variability is low, we approximately match the performance of the existing procedures.

References

[1] Al Fawzan, M., Haouari, M., 2005. A bi-objective model for robust resource-constrained project scheduling. International Journal of Production Economics 96, 175–187.

[2] Ballestín, F., 2007. When it is worthwhile to work with the stochastic RCPSP? Journal of Scheduling 10(3), 153–166.

[3] Ballestín, F., Leus, R., to appear. Resource-constrained project scheduling for timely project completion with stochastic activity durations. Production and Operations Management.

[4] Blazewicz, J., Lenstra, J., Rinnooy Kan, A., 1983. Scheduling subject to resource constraints: classification and complexity. Discrete Applied Mathematics 5, 11–24.

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

[6] Cho, J., Yum, B., 1997. An uncertainty importance measure of activities in PERT networks. International Journal of Production Research 35, 2737–2757.

[7] Chtourou, H., Haouari, M., 2008. A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling. Computers & Industrial Engineering 55, 183–194.

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

[9] Demeulemeester, E., Herroelen, W., 2002. Project Scheduling: A Research Handbook. Boston: Kluwer Academic Publishers.

[10] Drexl, A., 1991. Scheduling of project networks by job assignment. Management Science 37, 1590–1602.

[11] Elmaghraby, S., Fathi, Y., Taner, M., 1999. On the sensitivity of project variability to activity mean duration. International Journal of Production Economics 62, 219–232.

[12] Fernandez, A. A., Armacost, R. L., Pet-Edwards, J., 1996. The role of the non-anticipativity constraint in commercial software for stochastic project scheduling. Computers and Industrial Engineering 31, 233–236.

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

[14] Goldberg, D., 1989. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA.

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

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

[17] Gutierrez, G., Paul, A., 2001. Robustness to variability in project networks. IIE Transactions 33, 649–660.

[18] Hartmann, S., 1998. A competitive genetic algorithm for resource-constrained project scheduling. Naval Research Logistics 45, 733–750.

[19] Hartmann, S., Kolisch, R., 2000. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. European Journal of Operational Research 127, 394–407.

[20] Herroelen, W., Leus, R., 2004. Robust and reactive project scheduling: A review and classification of procedures. International Journal of Production Research 42(8), 1599–1620.

[21] Herroelen, W., Leus, R., 2005. Project scheduling under uncertainty, survey and research potentials. European Journal of Operational Research 165(8), 289–306.

[22] Holland, H., 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor.

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

[24] Kolisch, R., 1996. Efficient priority rules for the resource-constrained project scheduling problem. Journal of Operations Management 14, 172–192.

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

[26] Kolisch, R., Hartmann, S., 1999. Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. In: Weglarz, J. (Ed.), Project Scheduling. Recent Models, Algorithms and Applications. Kluwer, pp. 147–178.

[27] Kolisch, R., Hartmann, S., 2006. Experimental investigation of heuristics for resource-constrained project scheduling: An update. INFOR 174, 23–37.

[28] Kolisch, R., Padman, R., 2001. An integrated survey of deterministic project scheduling. Omega 29(3), 249–272.

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

[30] Lambrechts, O., 2007. Robust project scheduling subject to resource breakdowns. Ph.D. thesis, Katholieke Universiteit Leuven, Belgium.

[31] Leus, R., Herroelen, W., 2004. Stability and resource allocation in project planning. IIE Transactions 36(7), 667–682.

[32] Li, K., Willis, R., 1992. An iterative scheduling technique for resource-constrained project scheduling. European Journal of Operational Research 56, 370–379.

[33] Möhring, R., 2000. Scheduling under uncertainty: optimizing against a randomizing adversary. Lecture Notes in Computer Science 1913/2000, 651–670.

[34] Möhring, R., Radermacher, F., Weiss, G., 1984. Stochastic scheduling problems I – general strategies. ZOR – Zeitschrift für Operations Research, 28, 193–260.

[35] Neumann, K., Schwindt, C., Zimmermann, J., 2002. Project Scheduling with Time Windows and Scarce Resources. Springer.

[36] Özdamar, L., Ulusoy, G., 1996. A note on an iterative forward/backward scheduling technique with reference to a procedure by Li and Willis. European Journal of Operational Research 89, 400–407.

[37] Patterson, J., 1984. A comparison of exact procedures for solving the multiple constrained resource project scheduling problem. Management Science 30, 854–867.

[38] Pet-Edwards, J., Selim, B., Armacost, R. L., Fernandez, A., 1998. Minimizing risk in stochastic resource-constrained project scheduling. In: Proceedings of INFORMS Fall Meeting, Seattle, USA.

[39] Saliby, E., 1990. Descriptive sampling: a better approach to Monte Carlo simulation. Journal of the Operational Research Society 41, 1133–1142.

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

[41] Stork, F., 2000. Branch-and-bound algorithms for stochastic resource-constrained project scheduling. Tech. Rep. 702/2000, Technische Universität Berlin.

[42] Stork, F., 2001. Stochastic resource-constrained project scheduling. Ph.D. thesis, Technische Universität Berlin.

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

[44] Valls, V., Ballestín, F., Quintanilla, S., 2005. Justification and RCPSP: a technique that pays. European Journal of Operational Research 165, 375–386.

[45] Valls, V., Ballestín, F., Quintanilla, S., 2008. Hybrid genetic algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research 185, 495–508.

[46] Van de Vonder, S., Demeulemeester, E., Herroelen, W., Leus, R., 2005. The use of buffers in project management: The trade-off between stability and makespan. International Journal of Production Economics 97, 227–240.

[47] Wang, J., 2004. A fuzzy robust scheduling approach for product development projects. European Journal of Operational Research 152, 180–194.

[48] Wu, S. D., Byeon, E. S., Storer, R. H., 1999. A graph-theoretic decomposition of job shop scheduling to achieve scheduling robustness. Operations Research 47(1), 113–124.

[49] Xu, N., McKee, S. A., Nozick, L. K., Ufomata, R., 2008. Augmenting priority rule heuristics with justification and rollout to solve the resource-constrained project scheduling problem. Computers & Operations Research 35, 3284–3297.

[50] Yu, G., Qi, X., 2004. Disruption Management – Framework, Models and Applications. World Scientific, New Jersey.