J Sched (2018) 21:349–365
https://doi.org/10.1007/s10951-016-0505-x

New strategies for stochastic resource-constrained project scheduling

Salim Rostami[1,2] · Stefan Creemers[1,3] · Roel Leus[2]

Published online: 12 January 2017

Abstract We study the stochastic resource-constrained project scheduling problem or SRCPSP, where project activities have stochastic durations. A solution is a scheduling policy, and we propose a new class of policies that is a generalization of most of the classes described in the literature. A policy in this new class makes a number of a priori decisions in a preprocessing phase, while the remaining scheduling decisions are made online. A two-phase local search algorithm is proposed to optimize within the class. Our computational results show that the algorithm has been efficiently tuned toward finding high-quality solutions and that it outperforms all existing algorithms for large instances. The results also indicate that the optimality gap even within the larger class of elementary policies is very small.

Keywords Project scheduling · Uncertainty · Stochastic activity durations · Scheduling policies

1 Introduction

A *project* is a temporary endeavor to achieve clearly defined goals. *Project management* deals with the planning, organization, execution, monitoring (controlling) and closing of a project in order to attain the project's objectives (Project Management Institute 2013). A project entails a set of activities that have to be executed while respecting precedence constraints and resource and time limitations. *Project scheduling* belongs to the planning phase of project management, in which a schedule is developed that decides when to start and finish the activities in order to achieve the project's goals. Practical project management is usually confronted with scarceness of the resources available for processing the activities. Over the last decades, this has given rise to a large body of literature on resource-constrained project scheduling, with the so-called *resource-constrained project scheduling problem (RCPSP)* as a central problem.

In practice, some of the scheduling parameters may be uncertain. The exact duration of an activity, for instance, might not be known at the beginning of the project. One of the earliest sources for this observation is Malcolm et al. (1959). Similarly, the number of available resources is another parameter that may not be known before project execution. These uncertainties may be due to different sources, including estimation errors, unforeseen weather conditions, late delivery of some required resources, unpredictable incidents such as machine breakdown or worker accidents. For further motivation for the study of uncertainty in project scheduling, we refer to Lambrechts (2007), Yu and Qi (2004) and Wang (2004).

In the classic problem RCPSP, the goal is to find a schedule with minimum schedule length, or *makespan*. This is indeed by far the most frequently studied objective in the project management literature, although other objectives such as net present value and weighted earliness/tardiness have also received some attention. The *stochastic RCPSP* or *SRCPSP* is the optimization problem that results when the activity durations in RCPSP are modeled as stochastic variables.

✉ Roel Leus
roel.leus@kuleuven.be

Salim Rostami
s.rostami@ieseg.fr; salim.rostami@kuleuven.be

Stefan Creemers
s.creemers@ieseg.fr; stefan.creemers@kuleuven.be

[1] IESEG School of Management, Lille, France

[2] ORSTAT, KU Leuven, Leuven, Belgium

[3] Research Centre for Operations Management, KU Leuven, Leuven, Belgium

The uncertainty in processing times can have various causes, among which machine breakdowns (see Pinedo 2008). Since makespan is a function of the activity durations, the goal in SRCPSP is to minimize the *expected* makespan, and this will also be the objective in this article. All other parameters of RCPSP, in particular the resource requirements and availabilities, are assumed to be fully known at the time of scheduling. For examples of other objective functions in stochastic project scheduling, we refer to Leus (2003), Ben-david and Golany (2011), Bruni et al. (2011), Van de Vonder et al. (2008) and Deblaere (2010).

Based on the foregoing, SRCPSP can be seen as a generalization of the deterministic problem RCPSP. Since RCPSP is NP-hard (Blazewicz et al. 1983), the stochastic counterpart can also be expected to be intractable. Additionally, solution procedures for RCPSP may not be valid anymore; the main reason is that a solution to SRCPSP can no longer be represented as a single schedule (Stork 2001). Indeed, it needs to be decided for each possible scenario of activity durations when to start which activities, and so different schedules may result for different scenarios. A solution to SRCPSP is therefore a *policy*: a set of rules that prescribe how to dynamically schedule the activities in each possible scenario (Radermacher 1981). We will formalize this concept and discuss different policy classes in Sect. 2.

We distinguish three main strategies for tackling uncertainty in scheduling problems. Firstly, the decision maker may try to find a schedule that can tolerate minor deviations from the predicted values for the activity durations. This approach is typically called *robust* or *proactive* scheduling. The robustness of a schedule increases with its ability to absorb variability. For example, see Artigues et al. (2013). The resulting schedule is often called a *baseline* schedule, *predictive* schedule or *preschedule* for short.

The second strategy, *reactive scheduling*, iteratively "repairs" an initial schedule in order to adjust it to the realizations of the underlying stochastic variables, which are progressively observed during the execution of the project. This repair step focuses on rendering the schedule feasible again, minimizing the effect of disruptions and maintaining a good score on the initial objective (e.g., low makespan). In proactive scheduling, some simplifying assumptions are typically made about this repair step. In particular, it is to be noted that proactive and reactive scheduling are not mutually exclusive, but rather that they can be complementary. For more details, see Deblaere et al. (2011), Van de Vonder et al. (2005) and Chtourou and Haouari (2008).

The third type of strategy for executing a project in the context of SRCPSP is often called *stochastic scheduling*, and this is also the approach followed in this article. Here, no preschedule is built before the execution of the project, but starting from an empty initial schedule, a complete schedule (containing all activities) is constructed gradually as time progresses by means of a scheduling policy, exploiting the information that was gathered up until the current time (e.g., realized activity durations) as well as the a priori available information on the uncertainty of activity durations. The policies that we study are static (not modifiable during project execution), but decision making using a policy is dynamic, meaning that the policy typically responds differently in various scenarios, leading to different final schedules. Due to the absence of a baseline schedule, this approach is sometimes referred to as a *purely reactive* or *online* strategy. Scheduling policies can also be applied if a baseline schedule is used. This latter combination only appears rather rarely in the literature, however; see Leus and Herroelen (2004) for an example.

The main contributions of this work are fourfold: (1) A new class of policies is proposed that is a generalization of most of the classes described in the literature. (2) Our computational results show that our proposed procedure, optimizing within this new class, outperforms all existing algorithms, in the sense that it obtains higher-quality solutions with the same computational effort. (3) The results also indicate that the algorithm has been efficiently tuned toward finding high-quality solutions in the larger search space of the new class. In particular, for small instances, the optimality gap even within the larger class of elementary policies is very small—which is also a sign that the policy class itself contains very good elementary policies. (4) As an alternative to simulation-based evaluation of scheduling policies, we also examine an exact Markov chain evaluation subroutine. To this aim, a generalization of the Kulkarni–Adlakha Markov chain (Kulkarni and Adlakha 1986) is proposed to include start-to-start precedence constraints. Next to these four main contributions, we also describe a counterexample that shows that the class of elementary policies does not necessarily include a globally optimal policy within the class of static policies. Although we mainly evaluate our proposed algorithm based on the number of generated schedules during simulation, we also report our computation times as an alternative measure of computational effort, which can be useful for future works that are not merely based on simulation.

The remainder of this article is organized as follows: A number of definitions are provided in Sect. 2, together with a description of RCPSP, SRCPSP and scheduling policies. Section 3 outlines our ideas to extend the class of the so-called preprocessor policies, and solution evaluation is the subject of Sect. 4. A two-phase metaheuristic algorithm is proposed in Sect. 5 that allows us to find high-quality members within the newly proposed class of policies. Extensive computational results are reported in Sect. 6. A summary and some conclusions are given in Sect. 7.

2 Definitions

We first introduce the problem RCPSP in Sect. 2.1. Subsequently in Sect. 2.2, we provide a formal statement of SRCPSP. We then introduce different scheduling policies in Sect. 2.3, and in Sect. 2.4, we describe why the so-called elementary policies are not globally optimal.

2.1 The deterministic case

One of the inputs of an instance of RCPSP is a set of activities $N = \{0, \ldots, n\}$ with known deterministic durations $d_i \in \mathbb{N}$ for each activity $i \in N$. In SRCPSP, which is the central problem of this work, the assumption of known values for activity durations is relaxed, and the durations are modeled as random variables (see Sect. 2.2). All activities are executed without pre-emption, which means that once an activity is started, it is executed without interruption until its completion. Furthermore, K is a set of renewable resource types; each type $k \in K$ has a finite capacity a_k that remains unchanged throughout the project. Each activity $i \in N$ occupies r_{ik} units of each resource type $k \in K$ for the entire duration of its execution; we assume $0 \leq r_{ik} \leq a_k$. Activities 0 and n are dummy activities, serving as start and end of the project, with zero duration ($d_0 = d_n = 0$) and without resource usage ($r_{0k} = r_{nk} = 0$ for all $k \in K$).

A solution to (an instance of) RCPSP is a schedule, which is denoted by a vector $\mathbf{s} = (s_0, \ldots, s_n)$, in which s_i is the starting time of activity $i \in N$ in the schedule. Without loss of generality, we restrict starting times to integer values. The starting times have to respect a given set of precedence constraints, which are described by a directed acyclic graph $G(N, A)$, with A a partial-order relation on N (a binary relation that is transitive and irreflexive). Below, we will call such a relation A on N a *precedence relation*. Activity 0 is predecessor, and activity n is successor of all other activities.

We can now provide the following conceptual formulation of RCPSP:

minimize s_n
subject to

$$s_i + d_i \leq s_j \qquad \forall (i, j) \in A \qquad (1)$$

$$\sum_{i \in \mathcal{A}(s,t)} r_{ik} \leq a_k \qquad \forall t \in \mathbb{N}_0, \forall k \in K \qquad (2)$$

$$s_i \in \mathbb{N} \qquad \forall i \in N \qquad (3)$$

The constraint set (1) describes the precedence constraints between the activities. These are all of the finish-to-start (FS) type: The successor cannot be started before the predecessor is finished. Later in this text, we will also use start-to-start (SS) constraints: If activity pair $(i, j) \subset N \times N$ defines an SS-constraint, then this implies $s_i \leq s_j$. Equation (2) represents the resource constraints, where set $\mathcal{A}(\mathbf{s}, t)$ contains the activities that are in process during time period t (time interval $[t-1, t]$) according to schedule $\mathbf{s}$:

$$\mathcal{A}(\mathbf{s}, t) = \{i \in N : s_i \leq (t-1) \wedge (s_i + d_i) \geq t\}.$$

A schedule $\mathbf{s}$ that respects constraints (1)–(3) is called a *feasible* schedule.

Surveys of solution methods for RCPSP are provided in Demeulemeester and Herroelen (2002) and Neumann et al. (2006). While various exact methods have been described in the literature for obtaining optimal solutions for RCPSP, development of heuristic procedures has also received extensive attention as the computation time required for finding a guaranteed optimal solution becomes unacceptably large as the size of the instances grows. *Priority rules* are among the fastest of these heuristics; they build feasible schedules using a *schedule generation scheme (SGS)*. Such SGSs are important for this text because some of the scheduling policies for SRCPSP are derived from them. We discuss the two major types of SGS below. Both types take an activity (priority) list (i.e., a complete ordering of N) as input, and both stepwise add activities to a partial schedule.

1. The *parallel SGS* iteratively moves from one decision point to the next at which activities can be added (time incrementation). These decision points correspond with the beginning of the time horizon and with the completion times of already scheduled activities, and thus, at most n decision points need to be considered. At each decision point, each eligible activity is selected in the order of the priority list and it is scheduled on condition that no resource conflict arises. An activity is eligible if it is unscheduled and if all its predecessors according to A have been completed.
2. The *serial SGS* picks the next activity in the priority list in each iteration (activity incrementation), and the earliest possible starting time is assigned such that no precedence and resource constraints are violated. Consequently, exactly n iterations are needed to obtain a compete schedule.

It should be noted that the parallel SGS produces *non-delay* schedules, which are schedules in which activities cannot start earlier without delaying another activity even if activity pre-emption is allowed. The serial scheme, on the other hand, produces *active* schedules, which are schedules in which none of the activities can start earlier without delaying another activity without activity pre-emption. Any non-delay schedule is an active schedule, but the opposite is not true. While it can be shown that for each

RCPSP instance, there is at least one optimal active schedule, an optimal non-delay schedule does not necessarily exist. Additionally, for each active schedule there exists at least one activity list that will yield the schedule using the serial SGS, and similarly each non-delay schedule can be found via the parallel SGS. We refer to Kolisch (1996a, b) and Sprecher (2000) for details and applications.

2.2 The stochastic RCPSP

Contrary to RCPSP, in SRCPSP the duration of activity $i \in N$ is a random variable (r.v.) D_i, following a known probability distribution. If we denote the probability of event e by $Pr[e]$, then $\forall i \in N$, we have $Pr[D_i < 0] = 0$; we also assume $Pr[D_0 = 0] = Pr[D_n = 0] = 1$. The distributions may be fitted using historical data or experts' judgments; for a detailed discussion of the selection of a suitable distribution, see Schatteman et al. (2008), Al-Bahar and Crandall (1990), Chapman and Ward (2000), Dawood (1998) and Shtub et al. (2005). All durations are gathered in r.v. vector $\mathbf{D} = (D_0, D_1, \ldots, D_n)$.

A scheduling policy decides at each decision point which activities, if any, should be started. Decision points are typically the beginning of the time horizon and the completion time of each activity. At each decision point t, a policy can only use information that has become available up to t, together with a priori knowledge of the distributions. This restriction is called the *non-anticipativity constraint* (Stork 2001). Fernandez et al. (1996, 1998) note that some of the commercial project scheduling software available in the 1990s failed to take this constraint into account and consequently could produce misleading results.

A *realization* or *scenario* is a vector $\mathbf{d} = (d_0, d_1, \ldots, d_n)$, where each value d_i is a realization of D_i. Radermacher (1981) proposes to view a policy Π as a function $\mathbb{R}^{n+1}_{\geq} \rightarrow \mathbb{R}^{n+1}_{\geq}$ that maps scenarios $\mathbf{d}$ of activity durations to feasible schedules $\mathbf{s} = \Pi(\mathbf{d})$. Thus, for a given scenario $\mathbf{d}$, $[\Pi(\mathbf{d})]_i$ represents the starting time of activity i under policy Π; the makespan of schedule $\Pi(\mathbf{d})$ is then $[\Pi(\mathbf{d})]_n$. The goal of SRCPSP is to find a policy that minimizes $E[[\Pi(\mathbf{D})]_n]$, where $E[\cdot]$ is the expectation operator with respect to $\mathbf{D}$. This minimization is often restricted to a search over a specific class of policies. We will introduce a number of such classes that are of direct interest to this text in Sect. 2.3.

Most of the general concepts used in this section (such as scenarios, optimal policies, non-anticipativity) are not specific to stochastic scheduling only, but are borrowed from the literature on stochastic optimization. For further details, we refer to Wets (1989), Rockafellar and Wets (1991) and Escudero et al. (1993).

2.3 Scheduling policies

Scheduling policies may be optimized prior to project execution, with all the parameters decided and unchanged during the realization of the project. Such policies are referred to as *static* (*open-loop*) policies, and their class is denoted by $\mathcal{C}^{\mathrm{S}}$. Alternatively, a *dynamic* (*closed-loop*) policy runs an optimization routine for selecting the best set of starting activities at each decision point, based on the latest system information. While closed-loop policies are adaptive and more flexible than open-loop policies, they are generally perceived as being computationally very hard to manage. Consequently, work on optimization in this class of policies has remained very limited. In recent work, Li and Womer (2015) propose an approximate dynamic programming algorithm to find closed-loop policies for SRCPSP. Their computational results indicate that at the cost of significantly higher runtimes, the closed-loop algorithm outperforms open-loop algorithms for instances with asymmetric duration distributions, although open-loop policies remain superior for other instances.

In this work, we focus on open-loop policies. One particular subset of $\mathcal{C}^{\mathrm{S}}$ is the *elementary* policies (*EL-policies*), whose class is denoted by $\mathcal{C}^{\mathrm{EL}}$. An elementary policy starts jobs only at completion times of other activities and at time 0. Direct optimization over class $\mathcal{C}^{\mathrm{EL}}$ has only rarely been considered in the literature. In a recent article, Creemers (2015) models SRCPSP with phase-type distributions as a Markov decision process and proposes an exact algorithm for finding an optimal elementary policy. Unfortunately, an elementary policy does not always have a representation that is compact (polynomial) in the size of the instance, which limits optimization (either exact or heuristic) to small- and medium-sized instances. Below, we present some subclasses of elementary policies that have a more compact combinatorial structure.

2.3.1 RB-policies

Resource-based policies (*RB-policies*) are a direct extension of priority rules with the parallel SGS for RCPSP. An RB-policy takes an activity list as input and at each decision point tries to start each eligible activity in the order of the priority list. These policies are fast and easy to implement, but they have some disadvantages. In the function view of policies (Radermacher 1981), RB-policies are neither monotone nor continuous. One reason is that they suffer from the so-called *Graham anomalies* (Graham 1966): There is a possibility of increasing the project makespan when the duration of one or more activities is decreased. Additionally, even with deterministic processing times, there are instances for which no activity list yields an optimal schedule following an RB-policy. These observations are referred to by some researchers (e.g., Möhring 2000) as "unsatisfactory stability behavior" or "inadequate structural firmness" and have been invoked by some as a motivation to eliminate these policies from further study. We denote the class of RB-policies by symbol $\mathcal{C}^{\mathrm{RB}}$.

RB-policies, as well the other policy classes that will follow, are static: A policy is fully specified prior to project execution. As outlined in Sect. 2.2, a mapping is set up from scenarios to decisions (yielding schedules). Note, however, that this mapping is algorithmic in nature (progressively producing a schedule) and is not merely an analytic mathematical function.

2.3.2 AB-policies

Activity-based policies [*AB-policies*, also referred to as "job-based policies" (Stork 2001)]. These policies proceed similarly as RB-policies with the addition of the SS-constraints:

$$[\Pi(\mathbf{d};L)]_i \leq [\Pi(\mathbf{d};L)]_j, \qquad \forall\{i,j\} \subset N; i \prec_L j.$$

In words, for a given scenario $\mathbf{d}$, an RB-policy defined by an activity list L cannot start an activity j earlier than any of its predecessors i in L. Value $[\Pi(\mathbf{d};L)]_i$ is the starting time of activity i obtained from policy Π. Elimination of the SS-constraints yields a simple RB-policy with Graham anomalies, but the extra constraints improve the stability. AB-policies require more attention for the specification of the priority list. Define a *feasible* instance of SRCPSP to be an instance for which there exists a policy yielding a feasible schedule for every scenario. For a feasible instance, an RB-policy with an arbitrary input list will generate a feasible schedule for each scenario, but this is not always true for AB-policies. More precisely, for AB-policies, the activity list L should define a linear extension of the input order A, meaning that $i \prec_L j$ for each $(i,j) \in A$. AB-policies are derived logically from priority rules with the serial SGS for RCPSP, and this is why they are sometimes referred to as "stochastic serial SGS" (Ballestín 2007). This class is denoted by $\mathcal{C}^{\mathrm{AB}}$.

2.3.3 ES-policies

The class of earliest-start policies (ES-policies), denoted by $\mathcal{C}^{\mathrm{ES}}$, was first proposed by Radermacher (1981) and Igelmund and Radermacher (1983). For a binary relation E on N, let $T(E)$ denote its *transitive closure*, which is the (inclusion-)minimal transitive relation such that $T(E) \subseteq E$. A *forbidden set* $F \subset N$ is a set of activities that are pairwise not precedence related ($\nexists\{i,j\} \subset F : (i,j) \in A$), but that cannot be processed simultaneously due to the resource constraints ($\exists k \in K : \sum_{i \in F} r_{ik} > a_k$). A *minimal forbidden set* (*MFS*) is an inclusion-minimal forbidden set. We denote the set of MFSs for precedence relation E by $\mathcal{F}(E)$.

A policy $\Pi \in \mathcal{C}^{\mathrm{ES}}$ is parameterized by a set of activity pairs $X \subset (N \times N)\backslash A$ such that $\mathcal{F}(T(A \cup X)) = \varnothing$ and $G(A \cup X)$ is acyclic. Such a policy is said to "break" all MFSs, meaning that for each $F \in \mathcal{F}(A)$, there will be at least one pair $\{i,j\} \in F$ such that $(i,j) \in T(A \cup X)$: In effect, we are adding additional FS-constraints via X such that all potential resource conflicts are resolved beforehand. What remains is a new scheduling instance without resource constraints but with a denser precedence graph. This new instance is trivially solved for a given scenario $\mathbf{d}$ by starting each activity as early as possible, as follows:

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

and $[\Pi(\mathbf{d};X)]_0 = 0$. ES-policies are convex, monotone and continuous. Furthermore, Radermacher (1986) shows that any convex policy is an ES-policy. For further details and definitions, we refer to Radermacher (1985), Igelmund and Radermacher (1983), Stork (2001) and Radermacher (1981).

2.3.4 Preprocessor policies

The class $\mathcal{C}^{\mathrm{PP}}$ of *preprocessor policies* (*PP-policies*) was first introduced by Ashtiani et al. (2011). A PP-policy $\Pi \in \mathcal{C}^{\mathrm{PP}}$ is defined by a set of activity pairs $X \subset N \times N$ together with an activity list L, with $G(N, A \cup X)$ acyclic. Each pair in X induces an additional FS-constraint, and all remaining sequencing decisions are made dynamically during project execution by an RB-policy defined by L for the graph $G(N, A \cup X)$. Consequently, a PP-policy makes a number of a priori sequencing decisions before the project is started in a preprocessing step under the form of X. Note that this class is defined without specific attention to MFSs: an extra edge in X may or may not resolve resource conflicts, so that $0 \leq |\mathcal{F}(T(A \cup X))| \leq |\mathcal{F}(A)|$. In fact, the inclusion of edges that do not break any MFS may also have a beneficial effect on the expected makespan (Ashtiani et al. 2011).

2.3.5 Comparison

A major computational disadvantage of ES-policies, in comparison with policies using activity lists, is their dependence on computing all MFSs, the number of which grows exponentially with n. Stork (2001) concludes that, for large instances, using AB-policies is the only remaining alternative since they do not require the representation of resource constraints by MFSs. He considers RB-policies to be "inadequate" based on the statement that a minimal requirement for a policy is monotonicity and continuity (in view of policies as functions). We do not follow this argument: In line with Ashtiani et al. (2011), we conjecture that this absence of theoretical qualities hardly, if ever, constitutes an issue to a practical decision maker when the expected makespan is appropriately low. Let ρ^τ be the minimum expected makespan for policy class $\mathcal{C}^\tau$. Stork compares the minimum makespan for different classes of policies in the deterministic case and concludes that $\rho^{\text{ES}} = \rho^{\text{AB}} \leq \rho^{\text{RB}}$. For stochastic environments, on the other hand, he finds that the foregoing three policy classes are incomparable, providing examples with $\rho^1 < \rho^2$ as well as with $\rho^2 > \rho^1$ for each pair of classes $\{\mathcal{C}^1, \mathcal{C}^2\}$ out of $\mathcal{C}^{\text{RB}}$, $\mathcal{C}^{\text{AB}}$ and $\mathcal{C}^{\text{ES}}$.

The computational disadvantage of enumerating MFSs for extension of the precedence graph was circumvented by Ashtiani et al. (2011) by eliminating the requirement that extra precedence constraints break MFSs in the definition of PP-policies. Ashtiani et al. show that combining the SS-constraints that are inherent in AB-policies with the FS-constraints that come with ES-policies in the same way as PP-policies were formed, leads to a new class that is *not* a proper generalization of $\mathcal{C}^{\text{ES}}$, which is why they prefer to combine ES-policies with RB-policies rather than with AB-policies. Clearly, $(\mathcal{C}^{\text{RB}} \cup \mathcal{C}^{\text{ES}}) \subset \mathcal{C}^{\text{PP}}$: PP-policies combine the computational benefits and real-time dispatching features of $\mathcal{C}^{\text{RB}}$ with the structural stability and unconditional sequencing decisions of $\mathcal{C}^{\text{ES}}$. This does not mean, however, that $\mathcal{C}^{\text{PP}}$ automatically contains better solutions than $\mathcal{C}^{\text{AB}}$ or an extension of that class, although Ashtiani et al. do provide empirical evidence that PP-policies tend to be better than AB-policies, especially for medium- to high-variability duration distributions.

2.4 Elementary policies are not globally optimal

The recent literature on computational solutions for SRCPSP has always focused on optimizing over $\mathcal{C}^{\text{EL}}$ or over a subset of this class. It should be noted, however, that the class of elementary policies does not necessarily include the optimal policies with respect to all static policies. To the best of our knowledge, this observation has not been explicitly reported in the literature before, although the authors of earlier theoretical work in the 1980s were clearly implicitly aware of this, but did not pay much attention to it; see, for instance, Möhring et al. (1984) and (especially) Möhring and Radermacher (1989).

In the remainder of this text, we refer to an optimal static policy as a globally optimal policy. For the instance depicted in Fig. 1, the optimal elementary policy is dominated by a non-elementary (static) policy. Each node in the graph corresponds with one activity, with activity durations drawn from the finite set ω_i and resource requirement r_i for each $i \in N$. The network is the transitive reduction of the graph $G(N, A)$, meaning that transitive edges such as $(1, 4)$ are not included (although $(1, 4) \in A$). For activity 2, each of the two values in ω_2 has equal probability of 0.5 and the other activities only

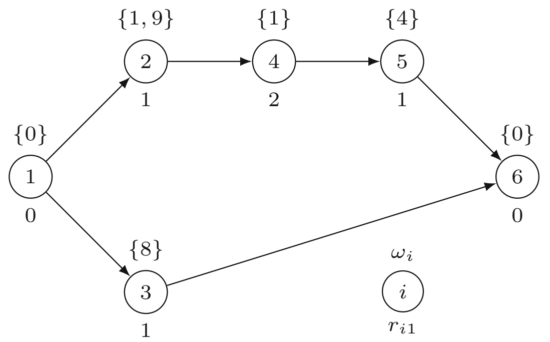

Fig. 1 A counterexample to show that elementary policies are not necessarily globally optimal

Table 1 Makespan for three policies, dependent on the duration of activity 2

D_2	$\Pi_{\text{EL}}^{\text{alt}}$	Π_{EL}^{*}	Π_1
1	10	13	10
9	18	14	14
Average	14	13.5	12

have one possible duration. There is one renewable resource type with availability $a_1 = 2$.

Table 1 summarizes the makespan values of three different policies. The optimal elementary policy Π_{EL}^{*} starts activities 2 and 3 in parallel, followed by activities 4 and 5 (in series). For information, we also include the details of an alternative elementary policy $\Pi_{\text{EL}}^{\text{alt}}$, which starts activity 3 together with activity 5 after the completion of activity 4. Finally, we also consider the following non-elementary policy Π_1, which starts activity 2 at time $t = 0$, and in which the decision when to start activity 3 is made at $t = 1$. If activity 2 is finished at $t = 1$, then activity 4 is started immediately and activity 3 will be started together with activity 5 afterward. Otherwise, activity 3 is started at $t = 1$. This policy is not elementary because when $D_2 = 9$, then the decision point $t = 1$ is not the completion time of any activity.

The table shows that Π_1 dominates its two elementary counterparts when it comes to expected makespan. Similar counterexamples can be constructed with continuous duration distributions, but these are typically less intuitive. In spite of this undesirable feature of elementary policies, the new policy class that we propose in Sect. 3 is also elementary because this will allow for a concise and structured description of the class, which makes it easier to develop an efficient optimization procedure.

3 Generalized preprocessor policies

3.1 Definition

We propose the new class of *generalized preprocessor policies* (*GP-policies*), denoted by $\mathcal{C}^{\text{GP}}$. A policy $\Pi \in \mathcal{C}^{\text{GP}}$ is defined by an activity list L together with two sets of activity pairs $X, Y \subset N \times N$. Each activity pair $(i, j) \in X$ defines an FS-constraint from activity i to activity j, while each $(i, j) \in Y$ induces an SS-constraint from i to j. The sets X and Y thus contain sequencing decisions made before the project starts. All remaining decisions are made dynamically during project execution by an RB-policy defined by L that respects all precedence constraints in $A \cup X \cup Y$. The reasons why the inclusion of SS-constraints (next to FS-constraints) might be beneficial for makespan minimization are explained in Sect. 3.3.

We say that GP-policy $\Pi(\mathbf{D}; L, X, Y)$ is *feasible* if for any realization of $\mathbf{D}$, the embedded parallel SGS (in the RB-policy) produces a feasible schedule given the constraints in L, X and Y. Theorem 1 states a necessary and sufficient condition for feasibility of a GP-policy.

Theorem 1 *A policy $\Pi \in \mathcal{C}^{\mathrm{GP}}$ is feasible if and only if $G(N, A \cup X \cup Y)$ is acyclic.*

Proof Assume there is a cycle in $G(N, A \cup X \cup Y)$. First, consider the case where all constraints (i, j) forming the cycle are of type SS. If there are sufficient available resources to start all activities of the cycle at the same time, then a feasible schedule might exist, but it cannot be produced by an RB-policy. This is due to the fact that using an SGS, activities of the priority list are scanned *one at a time*, and for each activity to be eligible, all of its predecessors should already be started. In other words, starting all activities of the cycle simultaneously is not considered by the SGS. The case with one or more FS-constraints (i, j) in the cycle can be discussed in a similar fashion, with the additional observation that a feasible schedule will certainly not exist in scenarios with $D_i > 0$.

Now assume that the policy $\Pi(\mathbf{D}; L, X, Y) \in \mathcal{C}^{\mathrm{GP}}$ is not feasible, so there exists at least one scenario for which the parallel SGS cannot produce a feasible schedule. Since we only consider instances where $\max_{i \in N} r_{ik} \leq a_k, \forall k \in K$, the resource constraints alone cannot cause this infeasibility, as one can always process all the activities consecutively, one at a time. Consider a scenario in which the SGS cannot produce a feasible schedule. Applying the SGS in this scenario, we gradually construct a partial schedule up to the point where the remaining activities cannot be scheduled. At the end of the latest finishing activity in this partial schedule, the SGS scans all the unscheduled activities one at a time in the order of L to see if any is eligible, but no such activity is found. Thus, for each unscheduled activity j, there is another activity i that has not yet been started and that needs to be either started (for an SS-constraint) or completed (for an FS-constraint) before j could be scheduled. In the graph induced by the nodes corresponding with the unscheduled activities, the edges corresponding with these SS-constraints and FS-constraints necessarily contain a cycle (since the activities cannot be linearly ordered). □

$$\begin{tikzcd}
\mathcal{C}^{AB} \arrow[drr] & & & & \\
\mathcal{C}^{ES} \arrow[dr] & & \mathcal{C}^{GP} \arrow[r] & \mathcal{C}^{EL} \arrow[r] & \mathcal{C}^{S} \\
& \mathcal{C}^{PP} \arrow[ur] & & & \\
\mathcal{C}^{RB} \arrow[ur] & & & &
\end{tikzcd}$$

Fig. 2 Hierarchy of different policy classes

3.2 Hierarchy

The hierarchy of the policy classes is graphically depicted in Fig. 2. An arc from one class to another means that the first class is included in the second one. The class of GP-policies encompasses $\mathcal{C}^{\mathrm{PP}}$ as well as $\mathcal{C}^{\mathrm{AB}}$, and therefore, the new class theoretically dominates $\mathcal{C}^{\mathrm{PP}}$ and $\mathcal{C}^{\mathrm{AB}}$. From a computational point of view, however, we need to verify whether a search procedure can be developed that is able to find solutions within the new class GP that are better than those found in the subclasses with the same computational effort, because the search area of the generalized class of policies is substantially larger than the search area of its subsets. The heuristic search procedure will be presented in Sect. 5, and computational results will be shown in Sect. 6.

3.3 Illustration and discussion

In essence, the functionality of additional FS-constraints in $\mathcal{C}^{\mathrm{ES}}$ is to "break" MFSs. Once all MFSs are resolved, an SGS is actually redundant since earliest possible start times can be obtained by means of *critical path method (CPM)* calculations, disregarding resource constraints altogether. In this case, FS-constraints have a clear advantage over SS-constraints in the sense that any single FS-constraint between two activities of an MFS resolves the MFS, while this is not necessarily true for SS-constraints. In $\mathcal{C}^{\mathrm{PP}}$, however, not all MFSs need to be resolved by the extra FS-constraints, and resource constraints cannot be neglected (hence, the priority list), and so the reasoning above for superiority of FS-constraints over SS-constraints does not hold. In any case, we know that the SS-constraints inherent in a serial SGS can sometimes help to find an optimal schedule for the deterministic RCPSP. Moreover, Ashtiani et al. (2011) have shown that FS-constraints in $\mathcal{C}^{\mathrm{PP}}$ that do not break any MFS can still help to achieve superior solutions. Below, we further illustrate the potential use of SS-constraints.

The first example, depicted in Fig. 3, shows a case where a feasible GP-policy outperforms an optimal member of $\mathcal{C}^{\mathrm{PP}}$.

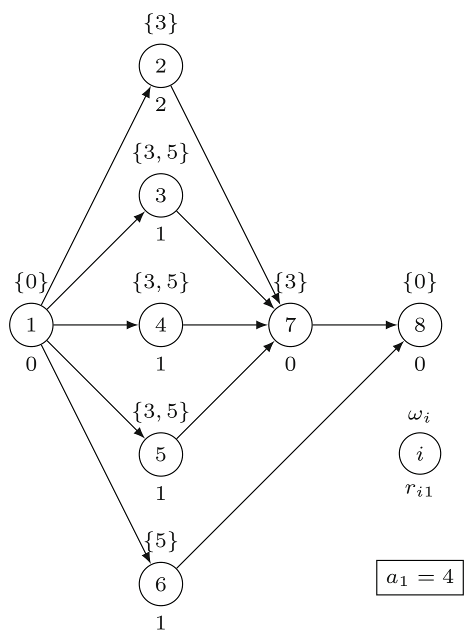

Fig. 3 A project instance where a feasible Π_{GP} outperforms an optimal Π^*_{PP}

In this example, each of the possible durations in ω_3, ω_4 and in ω_5 has equal probability 0.5. It is optimal to postpone activity 2 to be started not earlier than activities 3, 4 and 5, and also to postpone activity 6 to be started not earlier than activity 2. This policy assures that activity 2 is started following whichever activity with uncertain duration that finishes the earliest, and also that it is not postponed because of activity 6. Define $L_1 = (1, 3, 4, 5, 2, 6, 7, 8)$. For shorthand notation, throughout the remainder of the text, we will often identify the class of a policy by a subscript and omit the argument if there is no danger of confusion, so $\Pi_{\text{RB}}(L_1)$ is an RB-policy with parameter L_1. It can be shown that $\Pi_{\text{RB}}(L_1)$ and $\Pi_{\text{PP}}(L_1, \{(3, 6)\})$ are both optimal within their class. While $E[[\Pi_{\text{RB}}(L_1)]_n] = 10.00$ and $E[[\Pi_{\text{PP}}(L_1, \{(3, 6)\})]_n] = 9.75$, we have $E[[\Pi_{\text{GP}}(L_1, \emptyset, \{(2, 6)\})]_n] = 9.63$.

Secondly, from an optimization point of view, there are also indications that, with the same computational effort, we are more likely to find high-quality solutions within the larger search space of $\mathcal{C}^{\text{GP}}$ (which is empirically confirmed in Sect. 6). The example depicted in Fig. 4a shows a case where a given activity list L is only improvable via SS-constraints and not by FS-constraints. In this example, each of the durations in ω_5 and in ω_6 has equal probability 0.5. For any elementary policy, it is a dominant decision to postpone activity 3 to be started not earlier than activities 5 and 6. This ensures that activity 3 is started following the earliest finish from among activities 5 and 6. Table 2 compares different policies that achieve this. Define activity lists $L_1 = (1, 2, 4, 5, 6, 3, 7)$ and $L_2 = (1, 2, 3, 4, 5, 6, 7)$, and sets of additional SS-

Fig. 4 A project instance to demonstrate the importance of SS-constraints

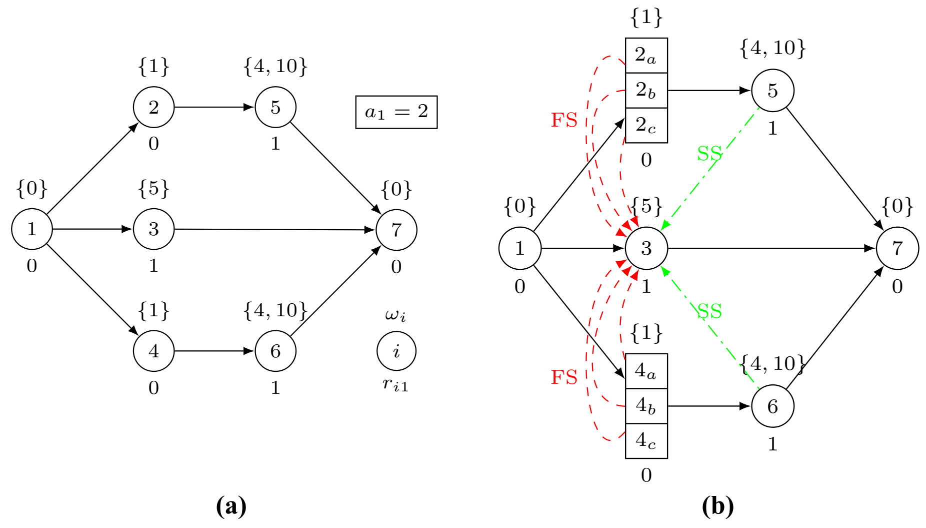

Table 2 Makespan for some RB-, PP- and GP-policies under different scenarios

(D_5, D_6)	$\Pi_{\text{RB}}(L_1)$	$\Pi_{\text{GP}}(L_1, \emptyset, Y_1)$	$\Pi_{\text{PP}}(L_1, X_1)$	$\Pi_{\text{RB}}(L_2)$	$\Pi_{\text{GP}}(L_2, \emptyset, Y_1)$	$\Pi_{\text{PP}}(L_2, X_1)$
(4,4)	9	10	10	9	10	9
(4,10)	15	11	11	15	11	15
(10,4)	11	11	11	11	11	11
(10,10)	15	16	16	15	16	16
Average	12.5	12	12	12.5	12	12.75

constraints $Y_1 = \{(5, 3), (6, 3)\}$ and FS-constraints $X_1 = \{(2, 3), (4, 3)\}$. It can be shown that $\Pi_{\mathrm{RB}}(L_1)$, $\Pi_{\mathrm{RB}}(L_2)$ and $\Pi_{\mathrm{PP}}(L_1, X_1)$ are each optimal within their class and that $\Pi_{\mathrm{PP}}(L_2, \emptyset)$ is the best PP-policy with list L_2. We observe that while both RB-policies are improvable via additional SS-constraints, only one of them ($\Pi_{\mathrm{RB}}(L_1)$) is also improvable with FS-constraints. This insight is important in view of the two-stage algorithm proposed Sect. 5, which selects an activity list in a separate stage prior to adding precedence constraints. If we worked with $\mathcal{C}^{\mathrm{PP}}$, selection of L_2 rather than L_1 in the first stage would then lead to a local optimum.

Finally, the example depicted in Fig. 4b presents a case where given an activity list that is improvable by both SS- and FS-constraints, it is easier to find the set Y_1 of SS-constraints rather than the set of FS-constraints required for equivalent performance. The example is an extension of the previous example where activities 2 and 4 are divided into three parallel activities, each with the same duration and resource requirements as before. Hence, the number of the predecessors of 5 and 6 is increased. Consequently, to adapt $\Pi_{\mathrm{PP}}(L_1, X_1)$ so as to stay equivalent with $\Pi_{\mathrm{GP}}(L_1, \emptyset, Y_1)$ (green arrows), X_1 must include all the FS-constraints from the sets of activities 2 and 4 to activity 3 (red arrows). In this example, from an optimization point of view, finding the set Y_1 (with $|Y_1| = 2$) with the same optimization effort is more likely than identifying X_1 (with $|X_1| = 6$).

4 Solution evaluation

Apart from the speed of convergence to optimality, the efficiency of optimization efforts for SRCPSP is also dependent on the accuracy and runtime for the evaluation of a policy. In line with the recent literature on SRCPSP, we will assess the quality of a scheduling policy based on the percentage difference between the expected makespan and the *critical path length (CPL)* using the average durations. We will test two different calculation methods for the expected makespan, namely using simulation and using a Markov chain.

Simulation is commonly used for expected makespan estimation. Stork (2001) uses a large set of scenarios (200) in each evaluation in order to increase the precision, while other researchers (for instance, Ballestín 2007 and Ashtiani et al. 2011) opt for a rather low number of replications (e.g., 10) in order to investigate more policies within the same simulation budget. The latter choice implies less accuracy for a given evaluation, but Ballestín (2007) shows that examining a larger set of policies is favorable for obtaining a better final outcome.

Creemers (2015) proposes an exact algorithm for SRCPSP with phase-type activity durations by making optimal decisions via dynamic programming in a Markov decision process; one of the prominent features of this procedure is efficient memory management for storing all required states of the decision process. Although this algorithm by itself is not computationally viable for large instances, we can derive from this procedure an exact evaluation subroutine that models the project execution as a Markov chain, and which can serve as an alternative to simulation. Some modifications are needed to the original procedure, for instance the inclusion of SS-constraints. More details on this Markov chain are provided in “Appendix” section.

5 A two-phase metaheuristic algorithm for $\mathcal{C}^{\mathrm{GP}}$

Metaheuristics are general algorithmic frameworks, often nature inspired, designed to solve complex optimization problems (Bianchi et al. 2009). In this section, we devise a two-phase metaheuristic that consists of a *greedy randomized adaptive search procedure* (abbreviated as *GRASP*) and a *genetic algorithm* (*GA*) to find high-quality $\Pi_{\mathrm{GP}}(L, X, Y)$.

GRASP, which was introduced by Feo and Resende (1995), consists of iterations made up from successive constructions of a greedy randomized solution and subsequent iterative improvements through local searches and self-learning techniques. Considering sequences as individuals, for example, each new sequence is divided into a number of subsequences. In order to fill each subsequence, a *reference* will be chosen. A reference may be to fill the elements of a subsequence randomly or according to another already-built randomly chosen sequence.

The population-based adaptive search procedure known as *GA* was introduced by Holland (1975) and is a heuristic search algorithm that mimics the process of natural evolution. 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 offspring per pair, followed by the mutation operator. Lastly, the next generation is created by invoking the selection operator that determines which individuals are carried over to the next generation and which ones are eliminated. We refer to Goldberg (1989) for a detailed discussion on GAs. The overall structure of the proposed two-phase metaheuristic is described in Sect. 5.1. Phase 1 is discussed in detail in Sect. 5.2, and Phase 2 is the subject of Sect. 5.3.

5.1 Global structure of the algorithm

Our search procedure consists of two phases. The first phase produces adequate activity lists by means of a GRASP, and in the second phase, a GA finds additional constraints to obtain a GP-policy with each list. Throughout the procedure, we distinguish between high-variability (HV) and low-variability

Algorithm 1 Overall structure

```
if HV then
    ElectList = RB-GRASP
    for $i = 1$ to NoList do
        Arc-Add-GA(ElectList($i$))
    end for
else if LV then
    ElectList = AB-GRASP
    for $i = 1$ to NoList do
        Arc-Remove-GA(ElectList($i$))
    end for
end if
Return the best solution found
```

settings (LV); our detailed criteria to distinguish between HV and LV are described in Sect. 6.1. This distinction is motivated by the observation that AB-policies (which impose numerous additional SS-constraints) are globally optimal for deterministic durations and also perform quite well for LV in general, whereas RB-policies have empirically been found to be far better for HV (see Ashtiani et al. 2011; Ballestín and Leus 2009). This is only logical, because the latter class retains more flexibility for managing unforeseen circumstances. Thus, for instances with HV, GRASP looks for a good RB-policy, whereas in LV, the first phase produces a good AB-policy. In both cases, the output is a set of activity lists, which is passed to the next phase. The overall structure of the proposed method is depicted in Algorithm 1: The set ElectList holds the best NoList solutions passed from Phase 1 to Phase 2.

5.2 Phase 1: activity lists

A general overview of the procedure RB-GRASP is shown in Algorithm 2, where the set CurSolPop is the current solution population. The LV-version of the function (AB-GRASP) is completely similar. The key element of the procedure is the BuildNewList function, which produces new individuals. A justification technique is employed in order to improve the quality of newly produced activity lists. A more detailed description of the main concepts follows.

Individuals and fitness Each individual is a precedence-feasible activity list L. An RB- or AB-policy Π then associates an expected makespan value $E[[\Pi(\mathbf{D}; L)]_n]$ with this list, which is the fitness indicator of L. This fitness value can be computed via exact methods such as a Markov chain, or estimated by means of simulation.

Building new lists The BuildNewList function builds new individuals. An overview of this function is provided in Algorithm 3. Firstly, each list is divided into multiple sublists. Each sublist is then filled according to a specific

Algorithm 2 RB-GRASP

```
CurSolPop = $\varnothing$
while TerminationCriterion not met do
    $L$ = BuildNewList( CurSolPop)
    s = $\Pi_{AB}(E[\mathbf{D}]; L)$
    s = Justification(s)
    $L$ = ScheduleToList(s)
    Compute $E[[\Pi_{RB}(\mathbf{D}; L)]_n]$
    if Cardinality(CurSolPop) $<$ PopSize$_1$ then
        CurSolPop = CurSolPop $\cup \{L\}$
    else if $L$ is better than the worst solution $L' \in$ CurSolPop then
        CurSolPop = (CurSolPop $\setminus \{L'\}) \cup \{L\}$
    end if
end while
ElectList = the NoList best solutions of CurSolPop
Return ElectList
```

Algorithm 3 BuildNewList(CurSolPop)

```
$i = 0$
FP $= 0$
while $i < n$ do
    if FP $= 0$ then
        reference = SelectReference( CurSolPop)
        if reference $\neq$ LFT or random then
            FP $\in$ [FP$_{\min}$, FP$_{\max}$]
        end if
    else
        FP $=$ FP $- 1$
    end if
    Select an activity $j \in E$ according to the reference
    $L(i) = j$
    $i = i + 1$
end while
Return the activity list $L$
```

reference. A random reference fills a sublist by randomly choosing activities from the set of eligible activities E. An eligible activity is an unselected activity for which all of the predecessors have already been selected. If LFT is chosen as a reference, biased random selection is applied, where activities have a higher chance of being selected if they have a small CPM-based latest finish time LFT. In order to make such selections, we incorporate *regret-based biased random sampling (RBRS)* such that:

$$\pi_j = \frac{\rho_j + 1}{\sum_{k \in E}(\rho_k + 1)}, \quad \forall j \in E,$$

where π_j is the selection probability of activity j and $\rho_j = \max_{k \in E}\{\mathrm{LFT}_k\} - \mathrm{LFT}_j$. The third reference type, pattern, is to choose activities from E according to another already-built activity list. The functioning period (FP) of a reference is the maximum number of times that the reference is allowed to be used before we choose a new one. For random or LFT patterns FP $= 1$, while for a pattern reference, it is chosen randomly from [FP$_{\min}$, FP$_{\max}$]. A new list is produced when all its sublists are filled.

Selecting reference Function SelectReference is used in order to assign references to sublists. To ensure sufficient diversity of the initial population, the reference type for the first $\mathsf{PopSize}_1$ (population size in Phase 1) solutions is restricted to random (with probability pRandom) and LFT (with probability $1-\mathsf{pRandom}$). For the next solutions, choosing pattern as a reference is possible (with probability $1 - \mathsf{pRandom} - \mathsf{pLFT}$). For this type, a reference activity list $L \in \mathsf{CurSolPop}$ is randomly chosen.

Justification In order to improve each new activity list, we apply a *double justification* technique (see Li and Willis 1992; Özdamar and Ulusoy 1996). Valls et al. (2005) show that justification is an effective technique to enhance RCPSP solutions without substantially more computational efforts. Both for HV and LV, a schedule $\mathbf{s}$ is first built by applying the serial SGS to the list over a single scenario with expected durations. 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 justified $\mathbf{s}$ is then reconverted into a list via ScheduleToList by ordering activities in non-decreasing order of their starting times. Preliminary experiments have indicated that using the parallel SGS for the justification of activity lists significantly decreases the diversity of the produced solutions in a population and leads to undesirable convergence to local optima.

5.3 Phase 2: additional precedence constraints

The second phase comprises a GA that finds sets of additional precedence constraints, which together with each activity list L in ElectList form a complete GP-policy. Dependent on the variability setting, the details of this phase differ slightly. For HV, we identify sets X and Y of additional FS- and SS-constraints to form $\Pi_{\text{GP}}(L, X, Y)$ that improves upon the RB-policy $\Pi_{\text{GP}}(L, \emptyset, \emptyset)$. In LV, on the other hand, starting from $\Pi_{\text{GP}}(L, \emptyset, \hat{Y})$ we look for a set $Y \subset \hat{Y}$ that leads to a good policy $\Pi_{\text{GP}}(L, \emptyset, Y)$, where $\hat{Y} = \{(i, j) | i \prec_L j\}$. The two variants of the algorithm are further elaborated in Sects. 5.3.1 and 5.3.2.

5.3.1 Phase 2 in HV

The goal is to find sets X and Y of additional precedence constraints such that

$$E[[\Pi_{\text{GP}}(\mathbf{D}; L, X, Y)]_n] < E[[\Pi_{\text{RB}}(\mathbf{D}; L)]_n].$$

For each $L \in \mathsf{ElectList}$, the GA produces an initial population and then iteratively builds new populations via crossover and mutation operators. Each population has size $\mathsf{PopSize}_2$.

Individuals An individual $Z = \{X, Y\}$ contains two unordered sets of activity pairs $(i, j) \notin A$. The individual is said to be feasible if and only if $G(N, A \cup X \cup Y)$ is acyclic.

Initial population Each initial population member contains between one and n_{pairs} activity pairs, with all cardinalities having equal probability. Note that members of subsequent generations can contain a number of elements that is not in $\{1, \ldots, n_{\text{pairs}}\}$. First, $\Pi_{\text{RB}}(E[\mathbf{D}]; L)$ is constructed, where at each decision point t, we encounter a set $N_t \subset N$ of activities that are either eligible to be started or are already in process. The set C_{SS} of candidate SS-constraints then contains each $(i, j) \in N_t \times N_t$ encountered at any decision point t for which the following criterion holds:

$$[\Pi_{\text{GP}}(E[\mathbf{D}]; L, \varnothing, \{(i, j)\})]_n < [\Pi_{\text{GP}}(E[\mathbf{D}]; L, \varnothing, \varnothing)]_n$$

and the same for C_{FS} with the following condition:

$$[\Pi_{\text{GP}}(E[\mathbf{D}]; L, \{(i, j)\}, \varnothing)]_n < [\Pi_{\text{GP}}(E[\mathbf{D}]; L, \varnothing, \varnothing)]_n.$$

The initial population is then constructed with individuals $Z = \{X, Y\}$ such that $X \subset C_{\text{FS}}$ and $Y \subset C_{\text{SS}}$, and more improving candidates have a higher selection probability.

Crossover The crossover operators for lists cannot be applied here; hence, we use a *uniform crossover* as follows. The two parents are randomly selected from the current population, with selection probability proportional to their quality. Each edge in the father is assigned to the son with probability p_{cross}; otherwise, it is added to the daughter. Each edge in the mother is analogously assigned to either daughter or son.

Mutation The mutation operator modifies some individuals in order to retain diversity in the population. Each solution Z is mutated with probability p_{mut}. If mutation occurs, then one randomly selected pair is removed with probability p_x; a random pair from $C_{\text{FS}} \backslash X$ and $C_{\text{SS}} \backslash Y$ is added to Z, otherwise.

Selection The selection operator is the same as in Phase 1: The solutions are ranked according to their objective value. The first $\mathsf{PopSize}_2$ solutions are then retained as the new generation.

5.3.2 Phase 2 in LV

In Phase 2 for LV, we search for a good policy $\Pi_{\text{GP}}(L, \emptyset, Y)$, so we only add SS-constraints and no FS-constraints. Each population again has size $\mathsf{PopSize}_2$, and the initial population is constructed as follows: $\mathsf{PopSize}_2 - 1$ solutions are generated similarly as in HV but with candidate set $\hat{Y}$, and one initial solution is the output of a greedy subroutine. For any $j \in N$, let $Y_j = \{(i, j) | i \in N, i \prec_L j\}$ be the set of all SS-constraints imposed on activity j. Starting from $Y' = \hat{Y}$, the greedy subroutine iteratively evaluates the condition

$$E[\Pi_{\mathrm{GP}}(\mathbf{D}; L, \varnothing, Y' \backslash Y_j)]_n] < E[\Pi_{\mathrm{GP}}(\mathbf{D}; L, \varnothing, Y')]_n]$$

for each $j \in N$ in order of list L. If the condition is satisfied, then Y' is updated as $Y' := Y' \backslash Y_j$. The algorithm stops when no further improvement is possible.

Similarly to Sect. 5.3.1, GA produces a final set Y by adding and/or removing constraints, and new generations are again iteratively constructed using similar crossover, mutation and selection functions. The algorithm halts when the simulation budget of the second phase is fully used.

6 Computational results

6.1 Experimental setup

All experiments have been performed on a personal computer with Intel i7-3770 CPU with 3.40 GHz clock speed and 8.00 GB of RAM. The algorithms are coded in Microsoft Visual Studio C++. Our main data set is the J120 instance set from the PSBLIB library, which was generated using the ProGen generator (Kolisch and Sprecher 1996). It includes 600 RCPSP instances with 120 non-dummy activities each. We will also use the J30 and J60 sets from the same library, which contain 480 instances with 30, resp. 60, activities each.

In line with Ashtiani et al. (2011), Ballestín and Leus (2009), Stork (2001), Fang et al. (2015) and Ballestín (2007), which are the most important works in the literature on SRCPSP that report computational results on large instances, we choose uniform, beta and exponential distributions for the activity durations. The expected activity durations are equal to the deterministic processing times $\mathbf{d}^* \in \mathbb{N}^{n+1}$ in the PSPLIB data sets. We use five different distributions to model the duration of an activity $i \in N$: two continuous uniform distributions with support $[d_i^* - \sqrt{d_i^*}; d_i^* + \sqrt{d_i^*}]$ and $[0; 2d_i^*]$; one exponential distribution with rate parameter $d_i^{*^{-1}}$; and two beta (generalized truncated) distributions with variance $d_i^*/3$ and $d_i^{*^2}/3$, both with support $[d_i^*/2; 2d_i^*]$. In the remainder of this text, we will refer to these five distributions as U1, U2, Exp, B1 and B2, respectively. The variances of these distributions are, in the same order, $d_i^*/3$, $d_i^{*^2}/3$, $d_i^{*^2}$, $d_i^*/3$ and $d_i^{*^2}/3$. Thus, U1 and B1 have relatively low variance, U2 and B2 have medium variability, and Exp displays high variability. Below, we will work with the HV-setting of our algorithm for the last three distributions, and with LV for U1 and B1. In both beta distributions, the parameter $\beta = 2\alpha$; for B1, we use $\alpha = (d_i^*/2) - (1/3)$, and for B2, we have $\alpha = (d_i^*/2) - (1/6)$.

Based on some preliminary experiments and on the findings of Ashtiani et al. (2011), we choose the probabilities $p_{\mathrm{cross}} = p_x = 0.5$ and $p_{\mathrm{mut}} = 0.05$, the population size in the first phase $\mathsf{PopSize}_1 = 40$ and the number of returned activity lists $\mathsf{NoList} = 1$. In the second phase, we set the maximum number of additional constraints in the initial population $n_{\mathrm{pairs}} = 7$, the population size $\mathsf{PopSize}_2 = 20$ and the parameters $\mathsf{FP}_{\mathsf{min}} = 1$ and $\mathsf{FP}_{\mathsf{max}} = 30$.

6.2 Policy evaluation

6.2.1 Simulation

The evaluation of the quality of an algorithm is based on the average percentage distance of $E[[\Pi(\mathbf{D})]_n]$ from the CPL with deterministic durations $\mathbf{d}^*$. The expected makespan is estimated via simulation or is obtained by means of a Markov chain evaluation subroutine (see Sect. 4). In most of the existing literature, scenarios are generated via simple Monte Carlo sampling, but Saliby (1990) observes that simple random sampling may lead to an imprecise description of known input distributions, which will increase the inaccuracy of simulations. In particular, since we intend to run only few generations, this problem might become severe. Consequently, in line with Ashtiani et al. (2011) and Ballestín and Leus (2009), we use descriptive sampling as a variance reduction technique, in which we use a random permutation of quantiles of the distribution at hand.

In the literature on SRCPSP, in order to compare different proposed algorithms despite the use of different computers, the optimization effort is controlled by allowing an equal simulation budget. More precisely, algorithm A is better than algorithm B if it finds better solutions with an equal number of generated schedules. In line with Ballestín and Leus (2009), Ballestín (2007), Ashtiani et al. (2011) and Fang et al. (2015), we use two upper bounds on the number of generated schedules, namely 5000 and 25,000. Ashtiani et al. (2011) observe that generating a schedule with a member from $\mathcal{C}^{\mathrm{RB}}$ or $\mathcal{C}^{\mathrm{PP}}$ requires approximately twice as much time as $\mathcal{C}^{\mathrm{AB}}$. Since GP-policies require the same computational requirement as PP-policies, we decide to adopt the following counting convention: One GP-policy in Phase 1 will be counted as 1 (schedule with RB-policy) + 2 (for applying the justification operator) + n_{sim} (number of simulations for evaluation) = $n_{\mathrm{sim}} + 3$. In Phase 2, each iteration of GA and each iteration of the greedy subroutine corresponds to n_{sim} scenarios. The number of iterations of the algorithm should be set based on this counting convention and the upper bound (5000 or 25,000) on the total number of schedule generations. In

Table 3 Comparing $E[[\Pi(\mathbf{D})]_n]$ for GP-H(SIM), GP-H(MC) and Exact in J30

Procedure	J30		
	Makespan	Gap (%)	CPU
GP-H(SIM)	75.22	0.91	0.07
GP-H(MC)	74.89	0.36	88.27
Exact	74.60	0.00	0.49

our implementation, we will evenly distribute the total budget of generated schedules among the two phases. Following Ashtiani et al. (2011) and Ballestín and Leus (2009), we opt for $n_{\text{sim}} = 10$.

6.2.2 *Comparison of simulation-based and exact policy evaluation*

We have examined the effect of replacing the simulation subroutine of our two-phase metaheuristic procedure, subsequently referred to as GP-H, with an exact evaluation subroutine based on a Markov chain approach (see Sect. 4). Both versions of the algorithm, denoted by GP-H(SIM) and GP-H(MC), have been applied to the J30 data set with exponential durations. The simulation budget in GP-H(SIM) is limited to 25,000 generated schedules; the number of policies examined by GP-H(MC) is the same as for GP-H(SIM). The results are then compared to the optimal elementary policies obtained by the exact algorithm proposed by Creemers (2015) (Exact). The details are provided in Table 3. The column labeled "gap" contains the percentage gap between optimal and heuristic makespan; runtimes are expressed in seconds. We observe that the Markov chain evaluation consumes significantly more CPU time (an increase by a factor of over 1000). The benefit, however, is that the average optimality gap is reduced to only one-third of its value with simulation evaluation, so there is a clear trade-off to be struck between runtime and quality of the solutions found. In the remainder of this text, we will only apply the simulation subroutine to estimate expected makespan because only in this way can comparisons be made with the published results for other procedures.

6.3 Comparison with other policies

Table 4 shows the comparison between our two-phase metaheuristic procedure (GP-H in the table) optimizing in $\mathcal{C}^{\text{GP}}$ with a GA for $\mathcal{C}^{\text{AB}}$ proposed by Ballestín (2007) (AB-GA), the GRASP algorithm for $\mathcal{C}^{\text{AB}}$ proposed by Ballestín and Leus (2009) (AB-GR), the two-phase GA for $\mathcal{C}^{\text{PP}}$ proposed by Ashtiani et al. (2011) (PP-GA) and the so-called estimation-of-distribution algorithm of Fang et al. (2015) (RB-EDA), for the J120 data set. We observe that GP-H outperforms all other algorithms in all five distributions. This supports the theoretical dominance of $\mathcal{C}^{\text{GP}}$ over $\mathcal{C}^{\text{AB}}$ and $\mathcal{C}^{\text{PP}}$ discussed in Sect. 3. Since the search space is significantly larger for $\mathcal{C}^{\text{GP}}$ than for the other policy classes, these results also indicate that the proposed two-phase algorithm has been efficiently tuned toward finding high-quality solutions in this large search space.

Table 4 Average percentage difference between the makespan and CPL for different algorithms for J120

Procedure	# Schedules	Distribution				
		U1	U2	Exp	B1	B2
AB-GA	5×10^3	51.49	78.65	120.22	–	–
	25×10^3	49.63	75.38	116.83	–	–
AB-GR	5×10^3	46.84	72.58	114.42	47.17	75.97
	25×10^3	45.21	70.95	112.37	45.60	74.17
PP-GA	5×10^3	48.86	58.91	76.03	49.01	58.82
	25×10^3	47.21	58.07	74.56	47.25	57.95
RB-EDA	5×10^3	47.29	56.54	72.50	47.65	58.29
	25×10^3	46.66	56.07	72.05	47.04	57.82
GP-H	5×10^3	46.71	55.95	71.71	46.87	55.95
	25×10^3	44.98	55.37	71.29	45.12	55.42

Table 5 Comparing the output of AB-GR and GP-H with optimal elementary policies in J30 and J60

Procedure	J30			J60		
	Makespan	Gap (%)	CPU	Makespan	Gap (%)	CPU
AB-GR	81.88	10.10	–	122.92	18.41	–
GP-H	75.22	0.91	0.07	112.13	1.19	0.30
Exact	74.60	0.00	0.49	110.59	0.00	831.58

The exact method of Creemers (2015) (Exact) for finding optimal elementary policies cannot be applied to this data set due to excessive memory usage: The largest instances that can be solved by the procedure have 30 to 60 activities. In Table 5, the results of the exact algorithm are compared with GP-H and AB-GR applied to J30 and J60, considering only exponential durations. The simulation budget in GP-H and AB-GR is limited to 25,000 generated schedules. Note that not all instances of J60 could be solved via Exact, so this comparison only includes the solved instances (227 out of 480). As before, "gap" is the percentage gap between optimal and heuristic makespan. Runtimes are expressed in seconds. We observe that the gap between the solutions obtained using our proposed algorithm and the optimal values is around 1% in both J30 and J60, while this gap for AB-GR is significantly higher.

Table 6 Runtimes (in seconds) for GP-H under different settings for J120

Procedure	# Schedules	Distribution				
		U1	U2	Exp	B1	B2
GP-H(LV)	5×10^3	92.7	93.4	94.8	207.6	201
	25×10^3	394.8	401.6	395.2	518.2	524.5
GP-H(HV)	5×10^3	339	331	301.6	450.9	432
	25×10^3	799.9	755.2	794.2	888.6	943.9

Note that we have assessed the performance of our algorithm compared to optimal elementary policies in this section, although elementary policies are not necessarily globally optimal (see Sect. 2.4). To the best of our knowledge, however, there are no publications in the literature that solve over a larger class of static policies with an expected makespan objective (although a limited number of studies have also looked into other than only elementary policies for other, so-called "non-regular," objectives, see, for instance, Buss and Rosenblatt 1997 and Bendavid and Golany 2011).

6.4 Runtimes

Although counting schedule generations is an accepted method for eliminating the impact of different computation devices (see Hartmann and Kolisch 2000, for instance), it is incompatible with approaches that are different from mere simulation-optimization (e.g., Creemers 2015; Li and Womer 2015). The goal of this section is therefore to report runtimes, as an alternative measure for computational effort, so that future researchers can evaluate their algorithms based on these times as well. Table 6 shows the runtimes of GP-H applied to J120 for all five distributions. GP-H(LV) refers to the GP-H algorithm with HV-setting (see Sect. 5.3.1), while GP-H(HV) is the LV-version described in Sect. 5.3.2.

The higher runtimes for GP-H(HV) are mainly due to the inclusion of preprocessing calculations in Phase 2 to create C_{SS} and C_{FS}. Also, due to the more time-consuming generation of random numbers, both versions of the algorithm are slightly slower for the beta distribution compared to U1, U2 and Exp.

6.5 Makespan as a function of computational effort

Recent work on SRCPSP using simulation-optimization methods has typically limited computational effort to 5000 and 25,000 schedules. In this section, we examine how these bounds affect the performance of the proposed algorithm. For this purpose, we have run GP-H on J120 (all five distributions) and J30 (only exponential distribution), with the budget on the schedule count varying from 10^3 to 200×10^3. Some of the parameters, including n_{sim}, PopSize and the number of iterations in GRASP and GA, are modified according to this budget.

Figures 5 and 6 summarize our findings. In all plots, the horizontal and vertical axis represent the number of schedules and the percentage difference between CPL and $E[[\Pi(\mathbf{D})]_n]$, respectively. In Fig. 6, the output of Exact and GP-H are compared for the exponential distribution. As depicted in the figures, in LV (U1 and B1) and also for the small instances (J30), a budget of 25×10^3 generated schedules seems to be sufficient to achieve the best performance of the heuristic algorithm. In HV (U2, B2 and Exp), on the other hand, a more extensive search yields a noticeably better final outcome. This suggests that extending the upper bound on the number of schedules can be useful for these distributions.

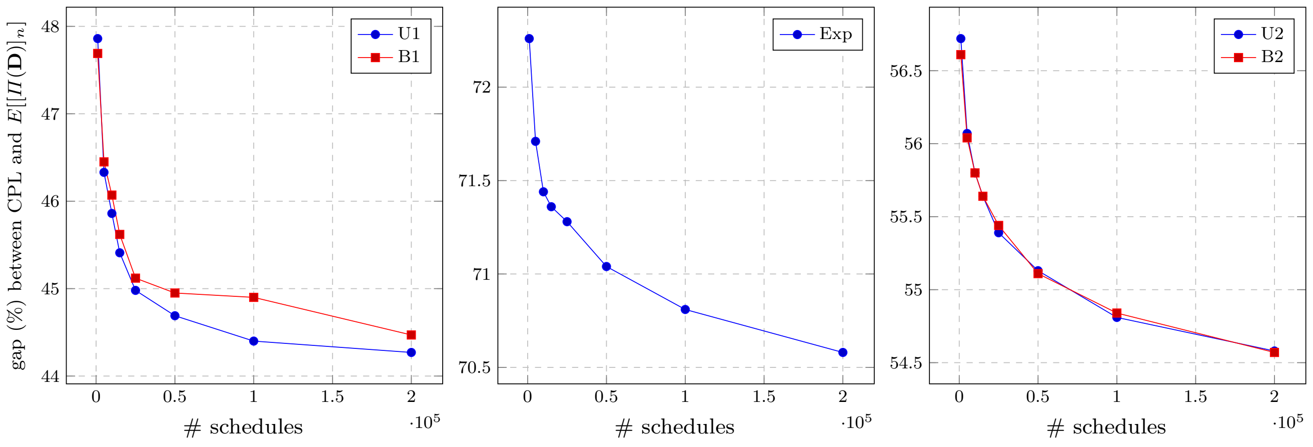

Fig. 5 Effect of the number of schedules on the performance of GP-H in J120

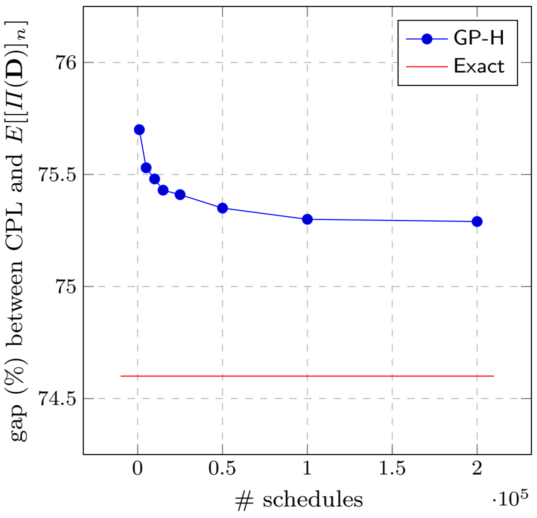

Fig. 6 Effect of the number of schedules on the performance of GP-H in J30 with exponential durations

7 Summary and conclusions

In this article, we have proposed the new class of generalized preprocessor policies (GP-policies) for the stochastic resource-constrained project scheduling problem (SRCPSP). The class of GP-policies is a generalization of the existing classes of RB-, AB-, ES- and PP-policies. A GP-policy makes a number of a priori scheduling decisions in a preprocessing phase under the form of additional precedence constraints, while the remaining decisions are made online by adhering to a priority list.

We have developed a two-phase algorithm for finding high-quality GP-policies. Our computational results show that the algorithm outperforms all existing procedures for large instances and that the algorithm has been efficiently tuned toward finding high-quality solutions in the larger search space of the new class. In addition, for small instances, the average optimality gap is very low although we compare with optimal elementary policies, which belong to an even larger class. This indicates that class of GP-policies by itself also contains very good elementary policies.

As an alternative to simulation-based evaluation of scheduling policies, we have also examined an exact Markov chain evaluation subroutine. To this aim, we have generalized the Kulkarni–Adlakha Markov chain in order to also include start-to-start precedence constraints. We find that the Markov chain evaluation is significantly more time-consuming, but also substantially increases the quality of the solutions found within the same number of evaluated solutions.

In this work, the additional precedence constraints (representing preprocessing decisions) are chosen by a local search algorithm with randomly evolving generations. For future work, it would be interesting to focus on adding more intelligence in the search for additional constraints, for instance by describing specific settings under which extra precedence constraints are particularly useful, or should be avoided.

Appendix: Exact evaluation of GP-policies for exponential distributions

In this appendix, we describe an exact evaluation procedure for the expected makespan of a feasible GP-policy $\Pi(L, X, Y)$, when each activity i has an exponentially distributed duration with rate parameter λ_i.

We use a Markov chain in which a state is represented by a pair (I, O), where I and O are the sets of idle and ongoing activities, respectively. The set F of finished activities is fully defined by given choices for I and O. In a state (I, O), an activity $i \in N$ is eligible to start if the following three conditions hold:

1. $i \in I$,
2. $j \in F$ for all j for which $(j, i) \in A$,
3. $r_{ik} \leq \left(a_k - \sum_{j \in O} r_{jk}\right)$ for all $k \in K$.

For a given state (I, O), let H denote the set of eligible activities and $W \subseteq H$ the set of activities to be started in that state (following the given policy $\Pi(L, X, Y)$). Activities $i \in H$ are considered in order of L for inclusion into W and are included if the following two conditions apply:

1. $j \in (O \cup F)$ for all j for which $(j, i) \in Y$,
2. $j \in F$ for all j for which $(j, i) \in X$.

If $|W| > 0$, then an immediate transition is made toward state $(I \backslash W, O \cup W)$. Otherwise (if W is empty), no activities are started and a transition takes place after completion of the first activity in O. The probability that an activity $i \in O$ finishes first equals $\lambda_i / \sum_{j \in O} \lambda_j$. The time until the first completion is exponentially distributed and has expected value $\left(\sum_{i \in O} \lambda_i\right)^{-1}$.

With each state (I, O), we associate a value function $G(I, O)$ that represents the expected time when state (I, O) is visited, and $\pi(I, O)$ denotes the probability that the state is visited. We stepwise update both values. If $W \neq \emptyset$, then $\pi(I \backslash W, O \cup W)$ is increased by $\pi(I, O)$ and $G(I \backslash W, O \cup W)$ is increased by $G(I, O)\pi(I, O)$. Otherwise ($W = \emptyset$), probability $\pi(I, O \backslash \{i\})$ is increased by $\pi(I, O)\lambda_i / \sum_{j \in O} \lambda_j$, and value function $G(I, O \backslash \{i\})$ is augmented with $\left(G(I, O) + \left(\sum_{i \in O} \lambda_i\right)^{-1}\right) \pi(I, O)\lambda_i / \sum_{j \in O} \lambda_j$.

Memory rather than computation time is typically the bottleneck when evaluating a Markovian PERT network. In our implementation, we have used techniques described in

Creemers et al. (2010) and Creemers (2015) to delete states from memory when they are no longer needed.

References

Al-Bahar, J. F., & Crandall, K. C. (1990). Systematic risk management approach for construction projects. *Journal of Construction Engineering and Management*, *116*, 533–546.

Artigues, C., Leus, R., & Talla Nobibon, F. (2013). Robust optimization for resource-constrained project scheduling with uncertain activity durations. *Flexible Services and Manufacturing Journal*, *25*(1–2), 175–205.

Ashtiani, B., Leus, R., & Aryanezhad, M. (2011). New competitive results for the stochastic resource-constrained project scheduling problem: Exploring the benefits of pre-processing. *Journal of Scheduling*, *14*(2), 157–171.

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

Ballestín, F., & Leus, R. (2009). Resource-constrained project scheduling for timely project completion with stochastic activity durations. *Production and Operations Management*, *18*, 459–474.

Bendavid, I., & Golany, B. (2011). Predetermined intervals for start times of activities in the stochastic project scheduling problem. *Annals of Operations Research*, *186*, 429–442.

Bianchi, L., Dorigo, M., Gambardella, L. M., & Gutjahr, W. J. (2009). A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, *8*(2), 239–287.

Blazewicz, J., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1983). Scheduling subject to resource constraints. *Discrete Applied Mathematics*, *5*, 11–24.

Bruni, M. E., Beraldi, P., Guerriero, F., & Pinto, E. (2011). A heuristic approach for resource constrained project scheduling with uncertain activity durations. *Computers & Operations Research*, *38*, 1305–1318.

Buss, A. H., & Rosenblatt, M. J. (1997). Activity delay in stochastic project networks. *Operations Research*, *45*(1), 126–139.

Chapman, C., & Ward, S. (2000). Estimation and evaluation of uncertainty: A minimalist first pass approach. *International Journal of Project Management*, *18*, 369–383.

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

Creemers, S. (2015). Minimizing the expected makespan of a project with stochastic activity durations under resource constraints. *Journal of Scheduling*, *18*(3), 263–273.

Creemers, S., Leus, R., & Lambrecht, M. (2010). Scheduling Markovian PERT networks to maximize the net present value. *Operations Research Letters*, *38*(1), 51–56.

Dawood, N. (1998). Estimating project and activity duration: A risk management approach using network analysis. *Construction Management and Economics*, *16*, 41–48.

Deblaere, F. (2010). *Resource constrained project scheduling under uncertainty*. Ph.D. thesis, Department of Applied Economics, KU Leuven, Belgium.

Deblaere, F., Demeulemeester, E., & Herroelen, W. (2011). Proactive policies for the stochastic resource-constrained project scheduling problem. *European Journal of Operational Research*, *214*(2), 308–316.

Demeulemeester, E., & Herroelen, W. (2002). *Project scheduling: A research handbook*. Boston: Kluwer Academic Publishers.

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

Fang, C., Kolisch, R., Wang, L., & Mu, C. (2015). An estimation of distribution algorithm and new computational results for the stochastic resource-constrained project scheduling problem. *Flexible Services and Manufacturing Journal*, *27*(4), 585–605.

Feo, T. A., & Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, *6*(2), 109–133.

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.

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.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.

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

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.

Holland, H. J. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press.

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

Kolisch, R. (1996a). Efficient priority rules for the resource-constrained project scheduling problem. *Journal of Operations Management*, *14*, 172–192.

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

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

Kulkarni, V. G., & Adlakha, V. G. (1986). Markov and Markov-regenerative PERT networks. *Operations Research*, *34*(5), 769–781.

Lambrechts, O. (2007). *Robust project scheduling subject to resource breakdowns*. Ph.D. thesis, KU Leuven, Belgium.

Leus, R. (2003). *The generation of stable project plans*. Ph.D. thesis, Department of Applied Economics, KU Leuven, Belgium.

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

Li, H., & Womer, N. K. (2015). Solving stochastic resource-constrained project scheduling problems by closed-loop approximate dynamic programming. *European Journal of Operational Research*, *246*(1), 20–33.

Li, K. Y., & Willis, R. J. (1992). An iterative scheduling technique for resource-constrained project scheduling. *European Journal of Operational Research*, *56*, 370–379.

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

Möhring, R. H. (2000). Scheduling under uncertainty: Optimizing against a randomizing adversary. In *Lecture Notes in Computer Science* (Vol. 1913/2000), pp. 651–670.

Möhring, R. H., & Radermacher, F. J. (1989). The order-theoretic approach to scheduling: The stochastic case. In R. Slowinski, J. Weglarz (Eds.), *Advances in Project Scheduling*, chapter III.4. Elsevier.

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

Neumann, K., Schwindt, C., & Zimmermann, J. (2006). *Project scheduling with time windows*. Berlin: Springer.

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.

Pinedo, M. L. (2008). *Scheduling: Theory, algorithms, and systems*. Berlin: Springer.

Project Management Institute. (2013). *A guide to the project management body of knowledge (PMBOK® Guide)*. Project Management Institute Inc.

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

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

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

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

Saliby, E. (1990). Descriptive sampling: A better approach to Monte Carlo simulation. *Journal of the Operational Research Society*, *41*, 1133–1142.

Schatteman, D., Herroelen, W., Van de Vonder, S., & Boone, A. (2008). A methodology for integrated risk management and proactive scheduling of construction projects. *Journal of Construction Engineering and Management*, *134*, 885–893.

Shtub, A., Bard, J. F., & Globerson, S. (2005). *Project management: Processes, methodologies, and economics*. New Jersey: Pearson Prentice Hall.

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

Stork, F. (2001). *Stochastic resource-constrained project scheduling*. Ph.D. thesis, Technische Universität Berlin.

Valls, V., Ballestín, F., & Quintanilla, S. (2005). Justification and RCPSP: A technique that pays. *European Journal of Operational Research*, *165*, 375–386.

Van de Vonder, S., Demeulemeester, E., & Herroelen, W. (2008). Proactive heuristic procedures for robust project scheduling: An experimental analysis. *European Journal of Operational Research*, *189*(3), 723–733.

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.

Wang, J. (2004). A fuzzy robust scheduling approach for product development projects. *European Journal of Operational Research*, *152*, 180–194.

Wets, R. J. B. (1989). *The aggregation principle in scenario analysis and stochastic optimization, volume F51 of Nato ASI Series* (pp. 91–113). Springer.

Yu, G., & Qi, X. (2004). *Disruption management—Framework, models and applications*. New Jersey: World Scientific.