New benchmark results for the stochastic resource-constrained project scheduling problem

Conference Paper · December 2015

DOI: 10.1109/IEEM.2015.7385637

3 authors, including:

Salim Rostami
University of Leuven
9 PUBLICATIONS **1** CITATION
SEE PROFILE

Stefan Creemers
IESEG School of Management
33 PUBLICATIONS **131** CITATIONS
SEE PROFILE

New Benchmark Results for the Stochastic Resource-Constrained Project Scheduling Problem

Roel Leus[1], Salim Rostami[2] and Stefan Creemers[3]
[1]ORSTAT, Faculty of Economics and Business, KU Leuven, Leuven, Belgium
[2]École Nationale Supérieure des Mines de Saint-Étienne, Saint-Étienne, France
[3]Management Department, IESEG School of Management, Lille, France
e-mail: roel.leus@kuleuven.be ; salim.rostami@emse.fr ; s.creemers@ieseg.fr

***Abstract*—In this article* 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 superset of most of the existing classes that are available in the literature. A policy in this new class makes a number of a-priori decisions in a pre-processing phase while the remaining scheduling decisions are made on-line. A two-phase local-search algorithm is proposed to find high-quality policies within the class. Our computational results indicate that the proposed procedure outperforms all existing algorithms for large instances.**

***Index Terms*—project scheduling, uncertainty, stochastic activity durations, scheduling policies.**

I. 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. 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 typically 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 the appearance of 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, for many reasons, some of the scheduling parameters may be uncertain. The exact duration of an activity, for instance, might not be known exactly at the beginning of the project. Similarly, the number of available resources is another parameter that may not be precisely 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, etc.

*This paper is an updated and extended version of the Master thesis of Salim Rostami [1], submitted to the Faculty of Economics and Business of KU Leuven in September 2013.

In the classic problem RCPSP the goal is to find a schedule with minimum schedule length or *makespan*. The *stochastic RCPSP* or *SRCPSP* is the optimization problem that results when the deterministic durations in RCPSP are replaced by stochastic variables, and the goal is then to minimize the *expected* makespan. All other parameters of RCPSP, in particular the resource requirements and availabilities, are assumed to be fully known at the time of scheduling. Thus, SRCPSP can be seen as a generalization of the deterministic problem RCPSP. Since RCPSP is NP-hard [2], 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 need no longer be a single schedule [3]. 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 decision rules that prescribe how to dynamically schedule the activities in each possible scenario [4]. We formalize this concept and discuss different policy classes in Section II.

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 activity durations. This approach is typically called *robust* or *proactive* scheduling. The resulting schedule is often called a *baseline* schedule, *predictive* schedule or *pre-schedule* 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 schedule. This repair step typically focuses on rendering the schedule feasible again, minimizing the effect of disruptions, and maintaining a good score on the initial objective. For more details, see [5]. 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 paper. Here no pre-schedule is built before the execution of the project, but a schedule is constructed gradually as time progresses by means of a scheduling policy. 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.

II. Definitions

We first introduce the deterministic problem RCPSP in Section II-A. Subsequently in Section II-B, we provide a formal statement of SRCPSP. We then introduce and briefly discuss different scheduling policies in Section II-C.

A. The deterministic case

One of the inputs of an instance of RCPSP is a set of activities $N = \{0, \ldots, n\}$ with precisely known 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 Section II-B). All activities are executed without preemption, 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; $d_0 = d_n = 0$ and $r_{0k} = r_{nk} = 0$ for all $k \in K$.

A solution to (an instance of) RCPSP is a schedule, which can be denoted by a vector $s = (s_0, \ldots, s_n)$, in which s_i is the starting time of the 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 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 a conceptual formulation of RCPSP:

$$\text{minimize } s_n$$

subject to

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

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

$$s_i \geq 0 \qquad \forall i \in N \qquad (3)$$

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. Further in this text, we also use start-to-start (SS) constraints: if $(i, j) \subset N \times N$ defines an SS-constraint, this means that $s_i \leq s_j$. Eq. (2) represents the resource constraints, where the set $\mathcal{A}(s,t) = \{i \in N : s_i \leq (t-1) \wedge (s_i + d_i) \geq t\}$ contains the activities that are in process during period t (time interval $[t-1, t]$) according to schedule s. A schedule s that respects constraints (1)–(3) is called a *feasible* schedule.

Surveys of solution methods for RCPSP are provided in [6] and [7]. While various exact methods have been described in the literature for obtaining optimal solutions for RCPSP, the development of heuristic procedures has also received extensive attention because the runtimes required for finding a guaranteed optimal solution become unacceptably high 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 list (a complete ordering of N), also called priority list, as input, which indicates the relative priority of the activities, and both stepwise add activities to a partial schedule.

1) The parallel SGS: this procedure 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 predecessors according to A have been completed).

2) The serial SGS: in each iteration, the next activity in the priority list is chosen (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 preemption was 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 preemption. Any non-delay schedule is an active schedule, but the opposite is not true. Although 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 yields the schedule using the serial SGS, and similarly each non-delay schedule can be found via the parallel SGS. We refer to [8] for details and applications.

B. The stochastic RCPSP

Contrary to RCPSP, in SRCPSP the duration of activity $i \in N$ is a random variable D_i with 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 suitable distributions see [9]. All durations are gathered in vector $D = (D_0, D_1, \ldots, D_n)$.

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* [3].

A *realization* or *scenario* is a vector $d = (d_0, d_1, \ldots, d_n)$, where each value d_i is a realization of D_i. Radermacher [4] proposes to view a policy Π as a function $\mathbb{R}_{\geq}^{n+1} \to \mathbb{R}_{\geq}^{n+1}$ that maps scenarios d of activity durations to feasible schedules $s = \Pi(d)$. Thus for a given scenario d, the quantity $[\Pi(d)]_i$ is the starting time of activity i under policy Π; the makespan of schedule $[\Pi(d)]$ is then $[\Pi(d)]_n$. The goal of SRCPSP is to find a policy that minimizes $E[[\Pi(d)]_n]$, where $E[\cdot]$ is the expectation operator with respect to D. This minimization is often restricted to a search over a specific class of policies. We introduce a number of such classes that are of direct interest to this text in Section II-C.

C. Scheduling policies

A policy that starts jobs only at completion times of other jobs or at time 0 is called *elementary*. 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-size instances [10]. Below we present subclasses of elementary policies that have a "nicer" combinatorial structure.

1) Resource-based policies (RB-policies): these 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 [4] RB-policies are neither monotone nor continuous. We denote the class of RB-policies by symbol $\mathcal{C}^{RB}$.

2) Activity-based policies (AB-policies): sometimes also referred to as "job-based policies" [3]. These policies proceed similarly as RB-policies with the addition of the SS-constraints

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

In words, for a given scenario d, an RB-policy defined by activity list L cannot start an activity j earlier than any of its predecessors i in L. Elimination of the constraint yields a simple RB-policy, 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 at least one feasible schedule under at least one scenario. For a feasible instance an RB-policy with an arbitrary input list generates 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" [11]. This class is denoted by $\mathcal{C}^{AB}$.

3) Earliest-start policies (ES-policies): this class, denoted by $\mathcal{C}^{ES}$, was first proposed in [12] and [4]. 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 $\Pi \in \mathcal{C}^{ES}$ is parameterized by a set of activity pairs $X \subset (N \times N) \setminus A$ such that $\mathcal{F}(T(A \cup X)) = \varnothing$ and $G(N, A \cup X)$ is acyclic. Such a policy is said to be "break" all MFSs, meaning that for each $F \in \mathcal{F}(A)$ there is at least one pair $\{i, j\} \in F$ such that $(i, j) \subset T(A \cup X)$: in effect, we are adding extra FS-type precedence 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, which is solved by starting each activity as early as possible, as follows:

$$[\Pi(D; X)]_j = \max_{(i,j) \in A \cup X} \{[\Pi(D; X)]_i + D_i\}, \quad \forall j \in N \setminus \{0\},$$

and $[\Pi(D; X)]_0 = 0$. ES-policies are convex, monotone and continuous. Furthermore, Radermacher [13] shows that any convex policy is an ES-policy.

4) Pre-processor policies (PP-policies): this class $\mathcal{C}^{PP}$ was first introduced in [14]. 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, 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 pre-processing step under the form of X, which contains extra edges for the precedence graph, such that $0 \leq |\mathcal{F}(T(A \cup X))| \leq |\mathcal{F}(A)|$. Note that this class is defined without specific attention to MFSs: the extra edges in X may or may not resolve a resource conflict. In fact, the inclusion of edges that do not resolve any MFS may also have a beneficial effect on the expected makespan [14].

5) Comparison: a major computational disadvantage of ES-policies in comparison to policies using activity lists is their dependence on computing all MFSs, the number of which is exponential with n. Stork [3] concludes that for large instances, using AB-policies is the only remaining alternative since these policies 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 [14], 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 different policy classes in the deterministic case and concludes that $\rho^{ES} = \rho^{AB} \leq \rho^{RB}$. For stochastic environments, on the other hand, he finds that the foregoing three 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 $\mathcal{C}^{RB}$, $\mathcal{C}^{AB}$ and $\mathcal{C}^{ES}$.

The computational disadvantage of enumerating MFSs for extension of the precedence graph is circumvented by Ashtiani et al. [14] 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 $\mathcal{C}^{ES}$, which is why they prefer to combine ES-policies with RB-policies rather than with AB-policies. Clearly, $(\mathcal{C}^{RB} \cup \mathcal{C}^{ES}) \subset \mathcal{C}^{PP}$: PP-policies combine the computational benefits and real-time dispatching features of $\mathcal{C}^{RB}$ with the structural stability and unconditional sequencing decisions of $\mathcal{C}^{ES}$. This does not mean, however, that $\mathcal{C}^{PP}$ automatically contains better solutions than $\mathcal{C}^{AB}$ 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.

III. Generalized pre-processor policies

We propose the new class of *generalized pre-processor policies* (*GP-policies*), denoted by $\mathcal{C}^{GP}$. A policy $\Pi \in \mathcal{C}^{GP}$ is defined by an activity list L together with two sets of activity pairs $X, Y \subset N \times N$. Each pair $(i,j) \in X$ defines an FS-constraint from activity i to 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 FS-constraints in $A \cup X$ as well as all the SS-constraints in Y. The reason why the inclusion of SS-constraints next to FS-constraints might be beneficial for makespan reduction, is that FS-constraints are more restrictive than SS-constraints and thus reduce flexibility in managing unpredicted disturbances. Thus there is a trade-off to be struck between pre-processing and on-line flexibility, and SS-constraints might be more suitable for this in a number of cases.

We say that GP-policy $\Pi(\cdot; L, X, Y)$ is *feasible* if for any duration realization, 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 $\Pi \in \mathcal{C}^{GP}$ is feasible if and only if $G(N, A \cup X \cup Y)$ is acyclic.*

We omit the proof here for lack of space.

The class of GP-policies encompasses $\mathcal{C}^{PP}$ as well as $\mathcal{C}^{AB}$ and theoretically, therefore, the new class dominates $\mathcal{C}^{PP}$ and $\mathcal{C}^{AB}$. 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 this new class that are better than those found in the subclasses with the same computational effort, because the search space of the generalized class of policies is substantially larger than its subsets.

IV. A two-phase meta-heuristic algorithm

A meta-heuristic is a high-level procedure that uses lower-level heuristics or search algorithms to find sufficiently good solutions for computationally hard optimization problems. We have devised a two-phase meta-heuristic for finding good GP-policies. In this search procedure, we make choices for the three parameters of a GP-policy, namely the activity list L, the FS-type precedence constraints corresponding with activity pairs in X, and the SS-constraints in Y.

Our meta-heuristic proceeds in two phases. In the first phase, a set of good order lists is constructed by means of a GRASP algorithm (Greedy Randomized Adaptive Search Procedure [15]), without considering extra precedence constraints. If the distribution of the processing time of the activities has medium to high variability, the GRASP looks for lists that define good RB-policies, whereas when the durations have low variability, the target of GRASP is to find lists that define good AB-policies. The reason for this differentiation is that these two policies compare quite differently depending on the variability of the durations (see [14], for instance). In the second phase of our search procedure, we use a genetic algorithm (GA [16]) to find suitable sets of SS-constraints and FS-constraints to include into X and Y, respectively. In the medium/high-variability setting, the initial solutions for the second phase have $X = Y = \varnothing$, whereas in case of low variability X is empty and Y contains all the activity pairs (i,j) for which i precedes j in L.

V. Computational results

In this section we summarize the results of our computational experiments to obtain high-quality scheduling policies. We first provide details on the experimental setup in Section V-A, and in Section V-B we compare the performance of the proposed algorithm for GP-policies with the best algorithms that are currently available in the literature (which optimize over other classes).

A. Experimental setup

All experiments were performed on a personal computer with Intel(R) Core(TM) i7-3770CPU with 3.40 GHz clock speed and 8.00 GB installed memory (RAM). The code was written and run in C++ using Microsoft Visual Studio Professional 2013 for Windows desktop under the Windows 7 x64 operating system. We evaluate our algorithm using J120[1], a standard dataset from PSPLIB [17] consisting of 600 RCPSP instances with 120 non-dummy activities each.

In line with the existing literature we examine Uniform, Beta and Exponential distributions for the activity durations. The expected durations are the deterministic processing times $d^* \in \mathbb{N}^{n+1}$ that are given in the J120 instances. We work with five different distributions: two continuous Uniform distributions with support $[d_i^* - \sqrt{d_i^*}; d_i^* + \sqrt{d_i^*}]$ and $[0; 2d_i^*]$; one Exponential distribution; and two Beta distributions with

[1] Available at http://www.om-db.wi.tum.de/psplib/

TABLE I
PERCENTAGE DIFFERENCE BETWEEN *CPL* AND $E[[\Pi(D)]_n]$ FOR DIFFERENT POLICIES Π.

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
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

variance $d_i^*/3$ and $d_i^{*^2}/3$, both with support $[d_i^*/2; 2d_i^*]$. Below, we 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 represent relatively low variance, U2 and B2 have medium variability, and Exp displays large variability. In both Beta distributions, $\beta = 2\alpha$; for B1, $\alpha = (d_i^*/2) - (1/3)$ and for B2, $\alpha = (d_i^*/2) - (1/6)$.

The evaluation of the quality of an algorithm is based on the percentage difference between $E[[\Pi(D)]_n]$ and the critical-path length *CPL* with deterministic durations d^*, and this averaged over all instances of J120. The expected makespan is estimated via simulation with 1000 replications for the final solution.

In order to compare computational results obtained on different computers, computational effort is typically measured by the number of generated schedules rather than by runtime [18]. In line with [11], [14], [19], we report our results for 5000 and 25000 schedules generated. Ballestín [11] observes that simulating one scenario with an AB-policy takes almost half the time needed for a serial SGS; he therefore counts a scheduling pass of an AB-policy as 0.5 while one run of a serial SGS is counted as 1. We opt for 10 as the number of replications in GA and GRASP for evaluation of an individual. This is a low number and thus the accuracy of the evaluation is low, but it allows us to generate and evaluate more solutions within the same computational effort.

B. Comparison with other policies

Table I presents the computational results of the GP-policies obtained by our two-phase meta-heuristic procedure ("GP-H" in the table) together with the results of the GA for $\mathcal{C}^{AB}$ proposed by Ballestín [11] ("AB-GA"), the GRASP algorithm for $\mathcal{C}^{AB}$ proposed by Ballestín and Leus [19] ("AB-GR"), the two-phase GA for $\mathcal{C}^{PP}$ proposed by Ashtiani et al. [14] ("PP-GA"), and the so-called "estimation-of-distribution" algorithm "EDA" of Fang et al. [20]. The exact method of Creemers [10] for finding optimal elementary policies cannot be applied to this instance set due to the excessive memory usage: the largest instances that can be solved by the procedure have between 30 and 60 activities.

We observe that GP-H outperforms all other algorithms for all distributions. This supports the theoretical dominance of GP-policies over AB and RB-policies discussed in Section III. Since the search space is significantly larger for GP-policies than for the other policy classes, these results also indicate that the proposed two-phase algorithm has been efficiently tuned towards finding good solutions in this large search space.

REFERENCES

[1] S. Rostami, "New benchmark results for the Stochastic Resource Constrained Project Scheduling (SRCPSP)," Thesis for obtaining the degree of Master of Advanced Business Studies, KU Leuven, 2013.

[2] J. Blazewicz, J. Lenstra and A. Rinnooy Kan, "Scheduling subject to resource constraints: Classification and complexity," *Discrete Applied Mathematics*, vol. 5, pp. 11–24, 1983.

[3] F. Stork, "Stochastic resource-constrained project scheduling," Ph.D. dissertation, Technische Universität Berlin, 2001.

[4] F.J. Radermacher, "Cost-dependent essential systems of ES-strategies for stochastic scheduling problems," *Methods of Operations Research*, vol. 42, pp. 17–31, 1981.

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

[6] K. Neumann, C. Schwindt and J. Zimmermann, *Project Scheduling with Time Windows*. Springer, ch. 15, 2006.

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

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

[9] C. Chapman and S. Ward, "Estimation and evaluation of uncertainty: A minimalist first pass approach," *International Journal of Project Management*, vol. 18, pp. 369–383, 2000.

[10] S. Creemers, "Minimizing the expected makespan of a project with stochastic activity durations under resource constraints," *Journal of Scheduling*, vol. 18, no. 3, pp. 263–273, 2015.

[11] F. Ballestín, "When it is worthwhile to work with the stochastic RCPSP?" *Journal of Scheduling*, vol. 10, no. 3, pp. 153–166, 2007.

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

[13] F.J. Radermacher, "Analytical vs. combinatorial characterizations of well-behaved strategies in stochastic scheduling," *Methods of Operations Research*, vol. 53, pp. 467–475, 1986.

[14] B. Ashtiani, R. Leus and M. Aryanezhad, "New competitive results for the stochastic resource-constrained project scheduling problem: Exploring the benefits of pre-processing," *Journal of Scheduling*, vol. 14, no. 2, pp. 157–171, 2011.

[15] T. Feo and M. Resende, "Greedy randomized adaptive search procedures," *Journal of Global Optimization*, vol. 6, no. 2, pp. 109–133, 1995.

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

[17] R. Kolisch and A. Sprecher, "PSPLIB – A project scheduling problem library," *European Journal of Operational Research*, vol. 96, pp. 205–216, 1997.

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

[19] F. Ballestín and R. Leus, "Resource-constrained project scheduling for timely project completion with stochastic activity durations," *Production and Operations Management*, vol. 18, pp. 459–474, 2009.

[20] C. Fang, R. Kolisch, L. Wang and C. Mu, "An estimation of distribution algorithm and new computational results for the stochastic resource-constrained project scheduling problem," *Flexible Services and Manufacturing Journal*, in press, DOI: 10.1007/s10696-015-9210-x.