KU LEUVEN

FACULTY OF ECONOMICS AND BUSINESS

# The preemptive stochastic resource-constrained project scheduling problem: an efficient globally optimal solution procedure

Creemers S.

KBI_1626

# The preemptive stochastic resource-constrained project scheduling problem: An efficient globally optimal solution procedure

Stefan Creemers

IESEG School of Management, Rue de la digue 3, 59000 Lille, France

KU Leuven, Research Center for Operations Management, Naamsestraat 69, 3000 Leuven, Belgium

s.creemers@ieseg.fr

We present a globally optimal solution procedure to tackle the preemptive stochastic resource-constrained project scheduling problem (PSRCPSP). A solution to the PSRCPSP is a policy that allows to construct a precedence- and resource-feasible schedule that minimizes the expected makespan of a project. The PSRCPSP is an extension of the stochastic resource-constrained project scheduling problem (SRCPSP) that allows activities to be interrupted. The SRCPSP and PSRCPSP both assume that activities have stochastic durations. Even though the deterministic preemptive resource-constrained project scheduling problem (PRCPSP) has received some attention in the literature, we are the first to study the PSRCPSP. We use phase-type distributions to model the stochastic activity durations, and define a new Continuous-Time Markov Chain (CTMC) that drastically reduces memory requirements when compared to the well-known CTMC of Kulkarni and Adlakha (1986). In addition, we also propose a new and efficient approach to structure the state space of the CTMC. These improvements allow us to easily outperform the current state-of-the-art in optimal project scheduling procedures, and to solve instances of the PSPLIB J90 and J120 data sets. Last but not least, if activity durations are exponentially distributed, we show that elementary policies are globally optimal for the SRCPSP and the PSRCPSP.

*Keywords:* Project management, continuous-time Markov chain, PSRCPSP

# 1 Introduction

The resource-constrained project scheduling problem (RCPSP) is one of the most widely studied scheduling problems. A solution to the RCPSP is a precedence- and resource-feasible schedule that minimizes the makespan of a project. A fundamental assumption of the RCPSP is that ongoing activities are non-preemptable. The preemptive resource-constrained project scheduling problem (PRCPSP) relaxes this assumption, and studies the RCPSP when activities are allowed to be interrupted. The literature on the PRCPSP is rather scarce. Kaplan (1988) was the first to study the PRCPSP, however, Demeulemeester and Herroelen (1996) have shown that the procedure proposed by Kaplan might not necessarily be optimal. Demeulemeester and Herroelen (1996) are the first to present an optimal procedure, and they use it to solve the instances of the Patterson data set. More recently, Damay et al. (2007) use a branch-and-bound algorithm to obtain the optimal solution to all 480 instances of the PSPLIB J30 data set. They are unable, however, to obtain optimal solutions for the instances of the PSPLIB J60 data set. Moukrim et al. (2015) also solve all instances of the PSPLIB J30 data set, and use a branch-and-price algorithm to obtain lower bounds for instances of the PSPLIB J60, J90, and J120 data sets. Because the PRCPSP is NP-hard in the strong sense (Ballestín et al. 2009), most researchers have focussed their efforts on heuristic procedures. Heuristic procedures are available from, among others, Damay et al. (2007), Vanhoucke and Debels (2008), Ballestín et al. (2008), Ballestín and Leus (2009), and Van Peteghem and Vanhoucke (2010).

All of the aforementioned studies assume that activity durations are known in advance (i.e., they are deterministic). In reality, however, activity durations are often uncertain (Herroelen and Leus 2004). The stochastic resource-constrained project scheduling problem (stochastic RCPSP or SRCPSP) studies the RCPSP when activity durations are stochastic. A solution to the SRCPSP is a policy that allows to construct a precedence- and resource-feasible schedule that minimizes the expected makespan of a project. Stork (2001) was the first to present an optimal solution procedure for the SRCPSP, and has solved 179 and 11 of the instances of the PSPLIB J30 and J60 data sets, respectively. More recently, Creemers (2015) was able to solve up to 303 instances of the PSPLIB J60 data set. Heuristic procedures have been developed in, among others, Golenko-Ginzburg and Gonik (1997), Tsai and Gemmill (1998), Ballestín (2007), Ballestín and Leus (2009), Ashtiani et al.

(2011), and Rostami et al. (2016). Most of these procedures adopt simple list policies (that try to execute activities in the order of a list). In this article, however, we adopt elementary policies (that allow decisions to be made at the end of activities, and at the start of the project). List policies are a subset of the class of elementary policies, and, in turn, elementary policies are a subset of the class of all policies (refer to Rostami et al. (2016) for a hierarchy of the different policy classes).

The preemptive SRCPSP (or PSRCPSP) is an extension of the SRCPSP that allows activities to be interrupted. In this article, we present a globally optimal procedure for solving the PSRCPSP. Our procedure uses a backward stochastic dynamic-programming (SDP) recursion to determine the expected makespan of a project that has preemptable activities and stochastic activity durations. We use acyclic phase-type (PH) distributions to model activity durations, and match the first two moments of the activity duration distributions. To the best of our knowledge, we are the first to study the PSRCPSP. In addition, we show that our approach significantly outperforms existing optimal procedures for other resource-constrained project scheduling problems. We are able to solve all instances of the PSPLIB J30 and J60 data sets with small computational effort. We also solve 196 instances of the PSPLIB J90 data set, and have even succeeded in solving some of the instances of the PSPLIB J120 data set. As such, we are the first to optimally solve instances of the PSPLIB J90 and J120 data sets.

Most of the literature on optimal stochastic project scheduling deals with Markovian PERT networks (i.e., PERT networks where activities have independent exponentially/PH-distributed durations). Markovian PERT networks have first been studied by Kulkarni and Adlakha (1986), who have used a continuous-time Markov chain (CTMC) to obtain the exact distribution of the earliest completion time of a project. The CTMC of Kulkarni and Adlakha (1986) has since then been used by, among others, Buss and Rosenblatt (1997), Sobel et al. (2009), Creemers et al. (2010), Creemers (2015), Creemers et al. (2015), and Gutin et al. (2015). In the CTMC of Kulkarni and Adlakha (1986), the state of the system is defined by three sets: the set of idle activities $I$, the set of ongoing activities $O$, and the set of finished activities $F$. Because activities are either idle, ongoing, or finished, the size of the state space has upper bound $3^n$, where $n$ is the number of activities in the project. Most of these states, however, do not satisfy precedence-and/or-resource constraints, and therefore, a strict partitioning of the state space is required. Most of the recent work on Markovian PERT networks uses uniformly directed cuts (UDCs) to structure the statespace. Although UDCs

allow to generate all feasible states, the identification of UDCs themselves is an NP-hard problem (Shier and Whited 1986).

In this article, we propose a new CTMC that only keeps track of the set of finished activities $F$. As a result, the size of the state space has upper bound $2^n$. In addition, we no longer use UDCs to structure the state space. Instead, we use two ordered arrays that not only reduce the computational effort required to generate/search the state space, but also reduce the number of states that are stored in memory at any one time. These improvements allow us to easily outperform the existing state-of-the-art in optimal project scheduling procedures. Last but not least, we show that, if activity durations are exponentially distributed, the globally optimal policy for the SRCPSP and the PSRCPSP is an elementary policy.

The remainder of this article is structured as follows. Section 2 presents some basic definitions and gives a brief problem statement. Section 3 defines the new CTMC, and Section 4 shows how the state space is structured. Section 4 also discusses the backward SDP recursion that is used to obtain the minimum expected makespan of a project. Section 5 explains how to incorporate PH-distributed activity durations. Section 6 provides a numerical example, and Section 7 holds the proofs that show that elementary policies are globally optimal if activity durations are exponentially distributed. Section 8 discusses the results of several computational experiments. Section 9 concludes.

## 2 Definitions and problem statement

A project is a network of activities that can be represented by a graph $G = (V, E)$, where $V = \{0, 1, \ldots, n\}$ is a set of nodes and $E = \{(i, j)|i, j \in V\}$ is a set of arcs. The nodes represent project activities, and the arcs connecting the nodes represent precedence relationships. Activities 0 and $n$ are dummy activities that represent the start and the completion of the project. The duration of an activity $i$ is a random variable $\tilde{p}_i$ that has expected value $\mu_i$. $\tilde{\mathbf{p}} = \{\tilde{p}_0, \tilde{p}_1, \ldots, \tilde{p}_n\}$ denotes the vector of the activity duration random variables. $p_i$ is a realization of $\tilde{p}_i$, and $\mathbf{p} = \{p_0, p_1, \ldots, p_n\}$ is a realization of $\tilde{\mathbf{p}}$. An activity $i$ can start when all of its predecessors are finished, and if sufficient resources are available. There are $K$ renewable resource types. The availability of each resource type $k$ is denoted by $R_k$. Each activity $i$ requires $r_{i,k}$ units of resource $k$, where $r_{0,k} = r_{n,k} = 0$, for all $k \in \mathscr{R} = \{1, 2, \ldots, K\}$.

A solution to the deterministic RCPSP is a schedule $S = \{S_0, S_1, \ldots, S_n\}$, where $S_i$ is the starting time of activity $i$, $S_0 = 0$, and $S_n$ represents the completion time of the project. In addition, define $\mathscr{A}(S,t) = \{i \in V : S_i \leq t \wedge (S_i + p_i) \geq t)$, the set of activities in schedule $S$ that are active at time $t$. A schedule $S$ is feasible if:

$$S_i + p_i \leq S_j \qquad \forall (i,j) \in E, \tag{1}$$

$$\sum_{i \in \mathscr{A}(S,t)} r_{i,k} \leq R_k \qquad \forall t \geq 0, \forall k \in \mathscr{R}, \tag{2}$$

$$S_i \geq 0 \qquad \forall i \in V. \tag{3}$$

The optimal schedule $S^*$ minimizes $S_n$ subject to Constraints 1–3.

The PRCPSP extends the RCPSP by allowing activities to be interrupted. The most general type of PRCPSP allows interruptions at any integer moment in time, and splits activities into series of phases that have unit duration. More formally, an activity $i$ is split into $p_i$ phases of unit duration that each require $r_{i,k}$ units of resource $k$, for all $k \in \mathscr{R}$. A solution to the PRCPSP is a schedule $\mathcal{S} = \{\mathcal{S}_0, \mathcal{S}_{1,1}, \mathcal{S}_{1,2}, \ldots, \mathcal{S}_{1,p_1}, \ldots, \mathcal{S}_n\}$, where $\mathcal{S}_{i,z}$ is the starting time of phase $z : 0 < z \leq p_i$ of activity $i$, $\mathcal{S}_0 = 0$, and $\mathcal{S}_n$ represents the completion time of the project. In addition, define, $\mathscr{A}(\mathcal{S},t) = \{i \in V : \exists\ 0 < z \leq p_i$ for which $\mathcal{S}_{i,z} = t)$, the set of activities in schedule $\mathcal{S}$ that are active at time $t$. A schedule $\mathcal{S}$ is feasible if:

$$\mathcal{S}_{i,p_i} + 1 \leq \mathcal{S}_{j,1} \qquad \forall (i,j) \in E, \tag{4}$$

$$\sum_{i \in \mathscr{A}(\mathcal{S},t)} r_{i,k} \leq R_k \qquad \forall t \geq 0, \forall k \in \mathscr{R}, \tag{5}$$

$$\mathcal{S}_{i,z} < \mathcal{S}_{i,(z+1)} \qquad \forall i \in V, \forall 0 < z < p_i, \tag{6}$$

$$\mathcal{S}_{i,z} \geq 0 \qquad \forall i \in V, \forall 0 < z \leq p_i. \tag{7}$$

The optimal schedule $\mathcal{S}^*$ minimizes $\mathcal{S}_n$ subject to Constraints 4–7.

The PSRCPSP is an extension of the PRCPSP that allows activities to have stochastic durations. Because activity durations are no longer known in advance, a solution to the PSRCPSP is a policy rather than a schedule. A policy $\Pi$ is a set of decision rules that defines actions at decision times. Decision times are typically the start of the project and the completion times of activities.

An action, on the other hand, corresponds to the start of a precedence-and-resource-feasible set of activities and/or the interruption of a subset of the ongoing activities. In addition, decisions have to respect the non-anticipativity constraint (i.e., a decision at time $t$ can only use information that has become available before or at time $t$). When executing a policy, activity starting times become known gradually (i.e., a schedule is constructed as time progresses). Consequently, a policy $\Pi$ may be interpreted as a function $\mathbb{R}_{\geq 0}^{n+1} \mapsto \mathbb{R}_{\geq 0}^{n+1}$ that maps realizations of activity durations $\mathbf{p}$ to vectors of feasible starting times $S(\mathbf{p};\Pi) = \{S(p_0;\Pi), S(p_1;\Pi), \ldots, S(p_n;\Pi)\}$ (see also Igelmund and Radermacher (1983), Möhring (2000), and Stork (2001)). For a given realization $\mathbf{p}$ and policy $\Pi$, $S_n(\mathbf{p};\Pi)$ denotes the makespan of schedule $S(\mathbf{p};\Pi)$. The objective of the PSRCPSP is to minimize $E(S_n(\mathbf{p};\Pi))$ over a class of policies, where $E(\cdot)$ is the expectation operator with respect to $\mathbf{p}$. Optimization over the class of all policies is computationally intractable. Therefore, we restrict our attention to the class of elementary policies that allows decisions to be made at the end of activities, and at the start of the project. In what follows, however, we show that, for the SRCPSP and the PSRCPSP, the globally optimal policy is an elementary policy if activity durations are exponentially/PH distributed.

## 3 A new CTMC

A project network with stochastic activity durations is often referred to as a PERT network, and a PERT network with independent exponentially-distributed activity durations is also referred to as a Markovian PERT network. Markovian PERT networks were first studied by Kulkarni and Adlakha (1986), who use a CTMC to determine the exact distribution of the completion time of a project where activities have exponentially-distributed durations. All existing work on Markovian PERT networks adopts the CTMC of Kulkarni and Adlakha (1986) to develop scheduling procedures (see e.g., Buss and Rosenblatt (1997), Sobel et al. (2009), Creemers et al. (2010), Creemers (2015), and Gutin et al. (2015)). In the CTMC of Kulkarni and Adlakha (1986), the state of the system is defined by three sets: the set of idle activities $I$, the set of ongoing activities $O$, and the set of finished activities $F$.

In this article, we propose a new CTMC that only keeps track of the set of finished activities $F$. In other words, our CTMC does not keep track of the set of ongoing activities. We can use the

set of finished activities, however, to determine the set of activities that are potentially ongoing. From this set of potentially ongoing activities, a policy then select the activities that are ongoing in each state of the system. This implies that the execution of an activity $i$ can be interrupted (i.e., whereas activity $i$ can be selected as a member of the set of ongoing activities at time $t$, it is not necessarily selected as a member at time $t + \Delta$). Note that, due to the memoryless property of the exponential distribution, the remaining work of an activity $i$ is exponentially distributed with rate parameter $\lambda_i$ at any time instance $t$ during which activity $i$ is ongoing (i.e., the remaining duration of activity $i$ is always the same). As a result, we do not have to keep track of the amount of work that has already been done when interrupting the execution of an activity. These properties make our CTMC especially suited for scheduling problems that allow the execution of activities to be interrupted.

More formally, let $F(t)$ denote all activities in $V$ that are finished at time $t$, and let $H(t)$ denote the set of activities that are potentially ongoing at time $t$. An activity $i$ is potentially ongoing at time $t$ if: (1) $i \notin F$ and (2) $j \in F$ for all $j$ for which $(i, j) \in E$. The starting and finishing conditions of the project are $F(0) = \emptyset$ and $F(t) = V$ for all $t \geq \omega$, where $\omega$ is the completion time of the project. Without loss of generality, we omit index $t$ when referring to sets $F(t)$ and $H(t)$.

The state of the system can be represented by the set of finished activities ($F$). Upon entry of state $(F) : F \neq V$, policy $\Pi$ determines the non-empty set of ongoing activities $O \subseteq H$. Of course, $O$ has to be a resource-feasible set of activities: $R_k \geq \sum_{i \in O} r_{i,k}$ for all $k \in \mathscr{R}$. The optimal policy $\Pi^\star$ selects the set of ongoing activities $O^\star$ from $H$ such that $G(\Pi^\star, F)$ is minimized, where $G(\Pi, F)$ is the value function that returns the expected time until completion of the project upon entry of state $(F)$ if policy $\Pi$ is adopted.

Given a set of ongoing activities $O$, the time until the first completion of an activity $i : i \in O$ is exponentially distributed with expected value $(\sum_{i \in O} \lambda_i)^{-1}$. The probability that activity $i : i \in O$ finished first equals $\lambda_i (\sum_{j \in O} \lambda_j)^{-1}$. Therefore, if policy $\Pi$ is adopted, the time until completion of the project upon entry of state $(F)$ equals:

$$G(\Pi, F) = \left( \sum_{i \in O} \lambda_i \right)^{-1} \sum_{i \in O} \lambda_i \left( \sum_{j \in O} \lambda_j \right)^{-1} G\left(\Pi, F \cup \{i\}\right). \tag{8}$$

The optimal subset of ongoing activities is given by:

$$O^{\star} = \underset{O \subseteq H}{\arg\min} \left( \sum_{i \in O} \lambda_i \right)^{-1} \sum_{i \in O} \lambda_i \left( \sum_{j \in O} \lambda_j \right)^{-1} G\left(\Pi^{\star}, F \cup \{i\}\right). \tag{9}$$

Unfortunately, determining the optimal set of ongoing activities is a non-linear problem that requires us to enumerate all resource-feasible subsets of $H$. Note, however, that several heuristics may be devised in order to determine a "good" set of ongoing activities. In addition, computational results show that, even if we enumerate all subsets in each state $(F)$, we still easily outperform all existing procedures that are based on the CTMC of Kulkarni and Adlakha (1986).

## 4 State-space structure and SDP-recursion

Most of the recent work on Markovian PERT networks uses UDCs to structure the state space of the CTMC of Kulkarni and Adlakha (1986) (see e.g., Creemers et al. (2010), Creemers (2015), and Gutin et al. (2015)). Although UDCs allow a strict partitioning of the state space, the identification of UDCs themselves is NP-hard (Shier and Whited 1986).

In this article, we no longer use UDCs to structure the state space. Instead, we use two ordered arrays that not only reduce the computational effort required to generate/search the state space, but also reduce the number of states that are stored in memory at any one time. A backward SDP-recursion is then used to determine the minimum expected makespan of a project. The recursion starts in state $(F) = V$, and completes upon reaching state $(F) = \emptyset$. The minimum expected makespan equals $\omega = G(\Pi^{\star}, \emptyset) = E\left(S_n\left(\mathbf{p}; \Pi^{\star}\right)\right)$. In addition, define $\mathbf{X}_i : \{(F) : |F| = i\}$, the array of states $(F)$ for which $i$ activities are finished, for all $i : 0 \leq i \leq n$. For each state $(F)$, we keep track of value function $G(\Pi^{\star}, F)$.

In continuous time, at most one activity can finish at any one time, and therefore, transitions can only be made from states in $\mathbf{X}_i$ towards states in $\mathbf{X}_{i+1}$. As such, in order to define the value function of a state $(F) \in \mathbf{X}_i$, we only need the value functions of all states in $\mathbf{X}_{i+1}$. In other words, at most two arrays of states (i.e., $\mathbf{X}_i$ and $\mathbf{X}_{i+1}$) have to be kept in memory at any one time. This results in a drastic reduction of memory requirements when compared to existing stochastic scheduling procedures that rely on UDCs to structure the state space.

We use Equation (8) to determine the value function of a state $(F) \in \mathbf{X}_i$, and use binary search to quickly lookup the value function of those states in $\mathbf{X}_{i+1}$ in which we end up after completion of one of the ongoing activities in $O^\star$, where $O^\star$ has been obtained using Equation (9). In order to use binary search: (1) we need to be able to identify states by means of some sort of lookup value, and (2) states need to be ordered based on this lookup value. Whereas Creemers et al. (2010) use tertiary numbers to serve as lookup values, we use binary numbers (i.e., each state $(F)$ can be represented by a binary number where bit $i$ reflects whether or not activity $i$ is finished). Upon initialization of array $\mathbf{X}_i$, new states are efficiently generated in the correct order.

After all value functions of all states $(F)$ in $\in \mathbf{X}_i$ have been determined, the memory used by the states in array $\mathbf{X}_{i+1}$ is allocated to store the value functions of all states in array $\mathbf{X}_{i-1}$. As a result, at most two arrays are kept in memory at any one time. Eventually, we obtain $G(\Pi^\star, \emptyset)$, the value function of state $(F) = \emptyset$, and have determined the minimum expected makespan of the project.

# 5 PH-distributed activity durations

PH distributions use exponentially-distributed building blocks (i.e., phases) to match any positive-valued distribution with arbitrary precision (see e.g., Neuts (1981) and Osogami (2005)). Because the phases themselves have exponentially-distributed durations, a project network with PH-distributed activity durations can easily be transformed into a Markovian PERT network (refer to Creemers (2015) for further details). The more phases, however, the more complex the Markovian PERT network becomes. In order to minimize the number of required phases, we use continuous-time PH distributions to match the first two moments of the duration distribution of an activity.

More formally, let $\nu_i = \sigma_i^2 \mu_i^{-2}$ denote the squared coefficient of variation (SCV) of the duration of activity $i$, where $\sigma_i^2$ is the variance of the duration of activity $i$. We define three cases: (1) $\nu_i = 1$, (2) $\nu_i < 1$, and (3) $\nu_i > 1$. In the first case, a single phase suffices, and the activity duration distribution can be approximated by means of an exponential distribution with rate parameter $\lambda_i = \mu_i^{-1}$. In the second case, a hypo-exponential distribution is used to model the duration distribution. The hypo-exponential distribution can be seen as a series of exponentially-distributed phases whose rate parameters are allowed to differ (i.e., the hypo-exponential distribution is a generalization of the

Erlang distribution). The SCV of the activity duration determines the number of phases that are required to approximate the duration distribution: $Z_i = \lceil \nu_i^{-1} \rceil$. To keep things simple, we assume that the first $Z_i - 1$ phases of the hypo-exponential distribution have i.i.d. exponential distributions with rate parameter:

$$\lambda_{i,1} = \lambda_{i,2} = \cdots = \lambda_{i,Z_i-1} = \frac{(Z_i - 1) - \sqrt{(Z_i - 1)(Z_i\nu_i - 1)}}{\mu_i(1 - \nu_i)}.$$

The last phase is exponentially distributed with rate parameter:

$$\lambda_{i,Z_i} = \frac{1 + \sqrt{(Z_i - 1)(Z_i\nu_i - 1)}}{\mu_i(1 - Z_i\nu_i + \nu_i)}.$$

In the third case, a two-phase Coxian distribution can be used to model the duration distribution of an activity. In this article, however, we only consider the first two cases, as an SCV larger than 1 is considered to be extremely variable. In fact, in the literature on stochastic project scheduling, the exponential distribution (with an SCV equal to 1) is often used as the most variable duration distribution (see e.g., Ballestín and Leus (2009), Ashtiani et al. (2011), and Rostami et al. (2016)).

Because each activity $i$ can be seen as a sequence of $Z_i$ phases that have exponentially-distributed duration, the project network can be transformed into a Markovian PERT network. As a result, the SDP-recursion introduced in Section 4 can once more be used to obtain the minimum expected makespan of the transformed Markovian PERT network.

## 6 Example

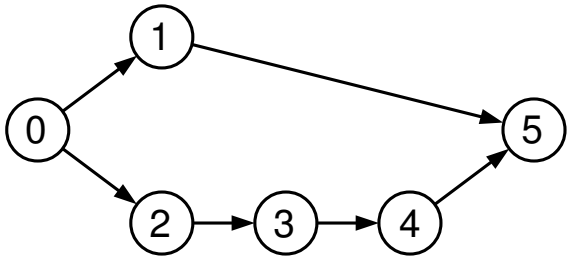
We use an example to illustrate the benefits of interrupting the execution of activities. The data of the example project is summarized in Table 1. Figure 1 visualizes the example project network. In the example, we assume there is a single resource (i.e., $K = 1$) that has an availability of 5 resource units (i.e., $R_1 = 5$). There are four non-dummy activities, and activity 1 can be executed in parallel with activities 2 and 4.

First, we observe what happens if the execution of an activity is not allowed to be interrupted. In this case, the optimal deterministic schedule has a makespan of 6 time units. If activity durations are exponentially distributed, however, the optimal expected makespan is 6.8 time units. Figures 2

Table 1: Data for the example project

| $i$ | $\mu_i$ | $r_{i,1}$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 3 | 3 |
| 2 | 2 | 2 |
| 3 | 2 | 5 |
| 4 | 1 | 2 |
| 5 | 0 | 0 |
| $R_1$ | 5 | |

Figure 1: Example project network




and 3 illustrate the optimal deterministic schedule and the optimal stochastic policy, respectively. The optimal stochastic policy starts activities 1 and 2 at the start of the project. We expect either activity 1 (with probability 0.4) or activity 2 (with probability 0.6) to finish after 1.2 time units. If activity 1 finishes first, activities 2, 3, and 4 are executed in series. If activity 2 finished first, on the other hand, activities 1, 3, and 4 are executed in series.

Next, we observe what happens if the execution of an activity is allowed to be interrupted. In this case, the optimal deterministic makespan can be reduced to 5 time units. If activity durations are exponentially distributed, the optimal expected makespan can be reduced to 6.35 time units. Figures 4 and 5 illustrate the optimal deterministic schedule and the optimal stochastic policy, respectively. The optimal stochastic policy starts activities 1 and 2 at the start of the project. If activity 1 finishes first, activities 2, 3, and 4 are executed in series. If, on the other hand, activity 2 finishes first, the execution of activity 1 is interrupted, and activity 3 is started. After completion of activity 3, activity 1 is resumed, and is executed in parallel with activity 4. In other words, the optimal policy tries to save time by executing activity 1 in parallel with activities 2 and/or 4.

Figure 2: Optimal schedule if activities have deterministic durations and their execution cannot be interrupted

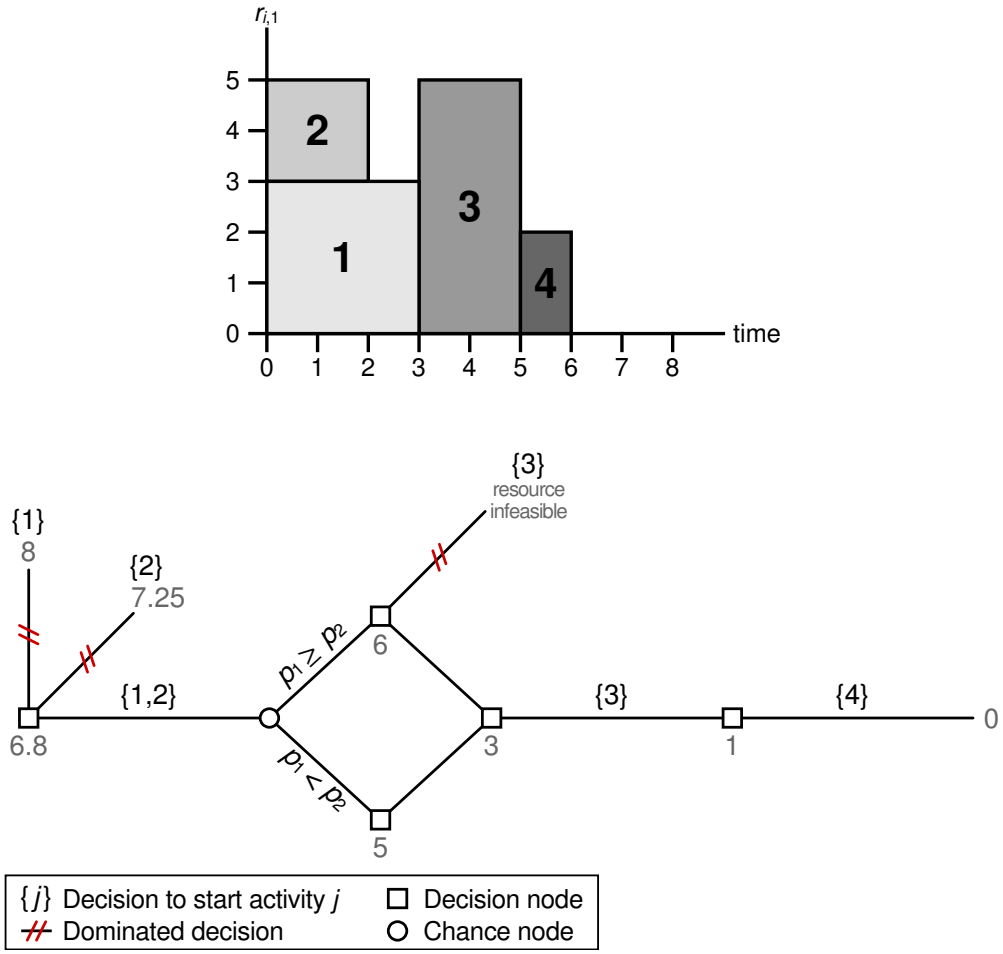



Figure 3: Optimal policy if activities have exponential durations and their execution cannot be interrupted

Figure 4: Optimal schedule if activities have deterministic durations and their execution cannot be interrupted

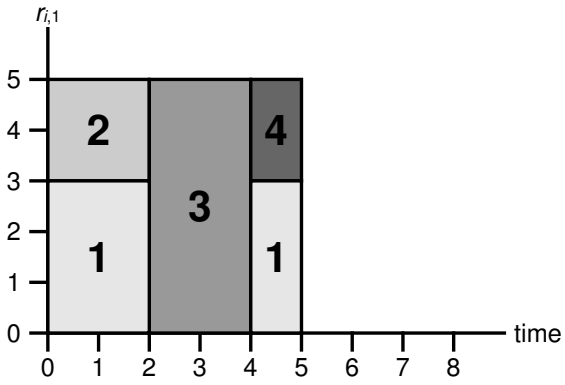
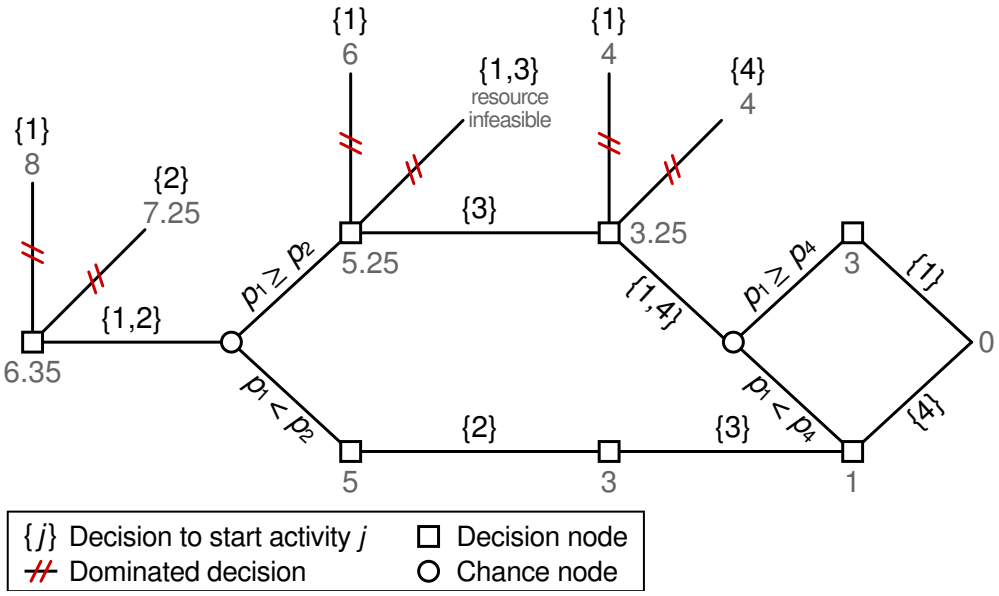


Figure 5: Optimal policy if activities have exponential durations and their execution can be interrupted




## 7 Globally optimal policies

For the SRCPSP, Rostami et al. (2016) use a counterexample to show that the globally optimal policy is not always an elementary policy (i.e., a policy that allows decisions to be made at the end of activities, and a the start of the project). The network and resource structure of their example is similar to the example that was introduced in Section 6. Instead of using exponentially-distributed activity durations, however, they assume deterministic durations for all activities except for activity 2. The duration of activity 2 can have two possible realizations (1 and 9), and each realization has equal probability. The other non-dummy activities have deterministic durations 8, 1 , and 4, respectively. The optimal elementary policy starts activities 1 and 2 in parallel, followed by activities 3 and 4 (in series), and has an expected makespan of 13.5 time units. The globally optimal policy, however, has an expected makespan of 12 time units, and starts activity 1 at $t = 0$, and decides whether or not to start activity 1 at $t = 1$. If activity 2 is still ongoing at time $t = 1$, activity 1 is started as well. If, on the other hand, activity 2 completes at time $t = 1$, activity 1 is postponed until after execution of activity 3, and is executed in parallel with activity 4. This policy is not elementary because $t = 1$ is not the completion time of an activity if activity 2 takes 9 time units. This example shows that non-elementary policies are useful to avoid a possible "lockdown"

of a resource (i.e., to avoid that a resource is unavailable at a time when it is needed).

Even though, for the general SRCPSP, elementary policies are not globally optimal, we show that, in the case of exponentially-distributed activity durations, the globally optimal policy is an elementary policy.

**Theorem 1.** *Elementary policies (that allow decisions to be made at the end of activities, and at the start of the project) are globally optimal for the SRCPSP if activity durations are exponentially distributed.*

*Proof.* The use of postponing the start of an activity is to obtain information that can be used to make better scheduling decisions. Without loss of generality, assume that as set of activities $O(t)$ is ongoing at time $t$. The start of activity $i$ is postponed until time $t + \Delta$. The goal of postponing activity $i$ is to obtain information on the activities in $O(t)$ that was not yet available at time $t$. Let $O(t + \Delta)$ denote the set of activities that are still ongoing at time $t + \Delta$. Due to the memoryless property of the exponential distribution, the remaining duration of all activities $j : j \in O(t + \Delta)$ is exponentially distributed with rate parameter $\lambda_j$. In other words, for the activities in $O(t + \Delta)$ no new information has become available at time $t + \Delta$. The only new information that has become available at time $t+\Delta$, is the set of activities that have finished in between time instance $t$ and time instance $t+\Delta$. There is no need, however, to postpone the start of activity $i$ in order to obtain this information. It suffices to only observe the instances in time at which an activity finishes. This is exactly what elementary policies do: allow to decisions to be made only at the completion times of activities (and at the start of the project). In fact, postponing activity $i$ will most likely result in a suboptimal solution, as there is a gap in between $t+\Delta$ and the time at which information becomes available (i.e., the completion times of activities in $O(t)$). □

**Theorem 2.** *Elementary policies (that allow decisions to be made at the end of activities, and at the start of the project) are globally optimal for the PSRCPSP if activity durations are exponentially distributed.*

*Proof.* The proof is similar to the proof of Theorem 1. There is no need to postpone the start of an activity $i$ until time $t = \Delta$/to interrupt an activity $i$ after a time $\Delta$, as no new information becomes available that would not have been available by only observing the completion times of activities. □

Table 2: Computational performance if activity durations are exponentially distributed (state-space sizes are expressed in millions of states)

| Data set | PAT | J30 | J60 | J90 |
|---|---|---|---|---|
| Instances in set | 110 | 480 | 480 | 480 |
| Instances solved | 110 | 480 | 480 | 196 |
| Avg # activities | 26 | 32 | 62 | 92 |
| Avg CPU time ($s$) | 0.003 | 0.025 | 8.6e3 | 214.7e3 |
| Max CPU time ($s$) | 0.016 | 0.393 | 252e3 | 6,023e3 |
| Min CPU time ($s$) | 0.000 | 0.002 | 0.0e3 | 0.243e3 |
| Avg state-space size | 0.001 | 0.010 | 71.69 | 681.5 |
| Max state-space size | 0.004 | 0.072 | 962.2 | 6,043 |
| Min state-space size | 0.000 | 0.001 | 0.082 | 21.68 |

Note that, if activity durations are PH distributed, the project network can be transformed into a Markovian PERT network where activities have exponentially-distributed durations (see Creemers (2015)). As a result, Theorems 1–2 also hold if activities have PH-distributed durations, and if decisions are allowed to be made at the end of a phase.

# 8 Results

In this section: (1) we assess the computational performance of our procedure, (2) we evaluate the usefulness of interrupting the execution of activities, and (3) we investigate the impact of activity duration variability. Our tests were performed on an Intel I5 3.3 GHz personal computer with 32 GB of RAM.

Table 2 and Figure 6 present the computational performance of our approach on the Patterson data set (Patterson 1984) and the well-known PSPLIB data sets (Kolisch and Sprecher 1996). From Table 2, it is clear that we are able to solve networks of up to 62 activities with small computational effort. We are also able to solve 196 instances of the J90 data set, and have even solved a few instances of the J120 data set. Because only a few instances of the J120 data set could be solved, we do not include them in the discussion of our results. Creemers (2015) concludes that memory, rather than computation time, is the main bottleneck when solving the SRCPSP. The same conclusion still holds when solving the PSRCPSP. However, we also observe that computation times start to become impractical when solving some of the more difficult instances of the J90 data set.

Figure 6: Computational performance on the Patterson and PSPLIB data sets if activity durations are exponentially distributed

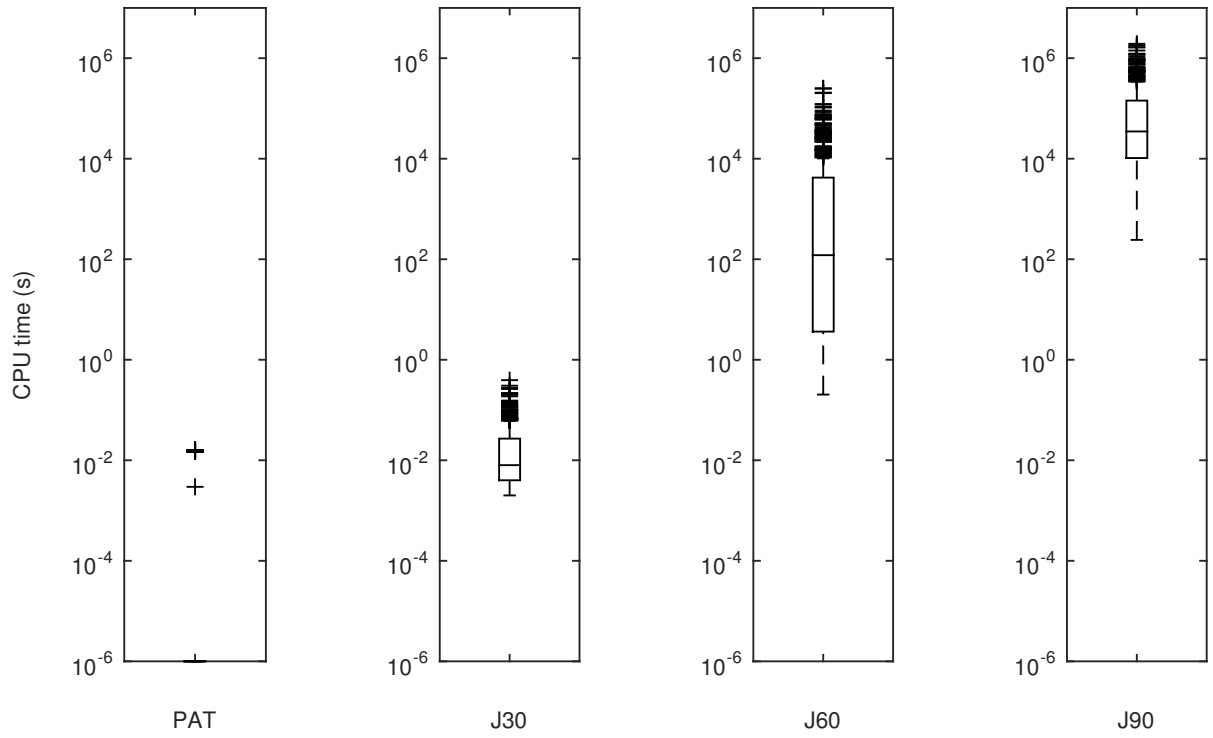


Table 3: Comparison of computational performance with approach of Creemers (2015) if activity durations are exponentially distributed (state-space sizes are expressed in millions of states)

| Data set | PAT | | J30 | | J60 | |
|---|---|---|---|---|---|---|
| Approach | SRCPSP | PSRCPSP | SRCPSP | PSRCPSP | SRCPSP | PSRCPSP |
| Instances in set | 110 | | 480 | | 480 | |
| Instances solved | 110 | | 480 | | 303 | |
| Avg # activities | 26 | | 32 | | 62 | |
| Avg CPU time ($s$) | 0.00 | 0.00 | 0.49 | 0.03 | 1,592 | 78.87 |
| Max CPU time ($s$) | 0.05 | 0.02 | 14.0 | 0.39 | 31,838 | 1,061 |
| Min CPU time ($s$) | 0.00 | 0.00 | 0.00 | 0.00 | 1.903 | 0.203 |
| Avg state-space size | 0.01 | 0.00 | 0.54 | 0.01 | 822.7 | 4.002 |
| Max state-space size | 0.14 | 0.00 | 11.4 | 0.07 | 4,257 | 32.85 |
| Min state-space size | 0.00 | 0.00 | 0.01 | 0.00 | 3.762 | 0.082 |
| Max % in memory | 28.2 | 18.0 | 32.7 | 19.8 | 45.52 | 12.71 |

Although we are the first to study the PSRCPSP, we can compare the computational performance of our approach with the performance of optimal procedures that study the SRCPSP (a problem that is "easier" than the PSRCPSP). In 2001, Stork (2001) was able to optimally solve 179 and 11 of the instances of the J30 and J60 data sets, respectively. More recently, Creemers (2015) also tackled the SRCPSP, and was able to solve up to 303 instances of the J60 data set. Table 3 further compares the difference in performance between our approach and the exact procedure of Creemers (2015). Note that, for the J60 data set, we can only compare the 303 instances that were solved in Creemers (2015). Even though the PSRCPSP is a more general problem, Table 3 clearly shows that our new approach significantly outperforms the optimal procedure of Creemers (2015). In fact, on average, we improve computational efficiency by a factor of 20.18, and memory efficiency by a factor of 205. These results are summarized in Table 4. This vast improvement in computational performance is in stark contrast to the findings of Demeulemeester and Herroelen (1996), who report a 33-fold increase in average computation time from the RCPSP to the PRCPSP (the respective deterministic counterparts of the SRCPSP and the PSRCPSP). Table 3 also reports the maximum percentage of states that are stored in memory at any one time. In contrast to the procedure of Creemers (2015), we no longer use UDCs, and as a result, memory requirements are drastically reduced.

Table 5 reports on the average decrease in optimal makespan if activities are allowed to be interrupted. The first part of the table deals with the RCPSP and the PRCPSP. Optimal solutions for the instances of the Patterson and J30 data sets have been obtained from Demeulemeester and

Table 4: Improvement in computational and memory efficiency when compared with the approach of Creemers (2015)

| Data set | PAT | J30 | J60 | TOT |
|---|---|---|---|---|
| CPU time improvement factor | 0.68 | 19.52 | 20.18 | 20.18 |
| Memory improvement factor | 12.1 | 53.60 | 205.6 | 205.0 |

Herroelen (1997) (for the RCPSP), and from Demeulemeester and Herroelen (1996) and Damay et al. (2007) (for the PRCPSP). For the instances of the J60 and J90 data sets, optimal solutions cannot be obtained directly, however, using lower and upper bounds they can often be determined indirectly. Many lower bounds and upper bounds (i.e., heuristic solutions) have been reported in the literature (refer to the PSPLIB website for the RCPSP, and to Moukrim et al. (2015) for the PRCPSP). If, for a given instance, lower and upper bound are the same, the optimal solution is known. In addition, we also adapted the branch-and-bound algorithm of Demeulemeester and Herroelen (1992) in order to obtain a number of optimal solutions (for the RCPSP) that have not yet been reported in the literature before. From the first part of Table 5, we observe that the benefit of interrupting activities decreases as the number of activities increases. This is not the case, however, when we look at the second part of the table, which deals with the SRCPSP and the PSRCPSP. In the second part of the table, we compare the optimal solutions of the SRCPSP and the PSRCPSP. The optimal solutions for the SRCPSP were obtained from Creemers (2015). It is clear that interrupting activities becomes more beneficial as the size of the network grows. The third part of Table 5, compares the reduction in optimal makespan for both the deterministic and the stochastic case, however, only considers the instances for which an optimal solution is known over all problems (i.e., the RCPSP, the PRCPSP, the SRCPSP, and the PSRCPSP).

All aforementioned results assume exponentially-distributed activity durations. Using PH distributions, however, we can approximate any positive-valued continuous distribution with arbitrary precision (Neuts 1981). Because PH distributions use exponentially-distributed building blocks, a Markovian PERT network can still be used, however, the size of the network increases with the number of phases in the PH distribution (Creemers 2015). The required number of phases depends on the variance of the activity duration distribution; the lower the variance, the more phases are required. Table 6 reports on the impact of variability on the computational performance of our

Table 5: Reduction in (expected) makespan if activities are allowed to be interrupted for deterministic and exponentially-distributed activity durations)

| Deterministic | PAT | J30 | J60 | J90 |
|---|---|---|---|---|
| Instances solved RCPSP | 110 | 480 | 421 | 395 |
| Instances solved PRCPSP | 110 | 480 | 383 | 299 |
| Instances shared over RCPSP & PRCPSP | 110 | 480 | 372 | 292 |
| Avg % reduction over RCPSP & PRCPSP | 0.78 | 1.34 | 0.55 | 0.03 |
| Stochastic | PAT | J30 | J60 | J90 |
| Instances solved SRCPSP | 110 | 480 | 303 | 0 |
| Instances solved PSRCPSP | 110 | 480 | 480 | 196 |
| Instances shared over SRCPSP & PSRCPSP | 110 | 480 | 303 | 0 |
| Avg % reduction over SRCPSP & PSRCPSP | 1.00 | 1.55 | 1.93 | NA |
| Deterministic & stochastic | PAT | J30 | J60 | J90 |
| Instances shared over all problems | 110 | 480 | 234 | 0 |
| Avg reduction over RCPSP & PRCPSP | 0.78 | 1.34 | 0.68 | 0 |
| Avg % reduction over SRCPSP & PSRCPSP | 1.00 | 1.55 | 1.26 | NA |

Table 6: Computational performance for different values of SCV when solving the J30 instances of the PSPLIB data set (state-space sizes are expressed in millions of states)

| Activity duration SCV | 1/2 | 1/3 | 1/4 |
|---|---|---|---|
| Number of phases | 2 | 3 | 4 |
| Avg # activities | 62 | 92 | 122 |
| Instances solved (PSRCPSP) | 480 | 480 | 480 |
| Instances solved (SRCPSP) | 480 | 421 | 358 |
| Avg CPU time ($s$) | 4.704 | 182.96 | 4,571 |
| Max CPU time ($s$) | 199.9 | 11,660 | 470e3 |
| Min CPU time ($s$) | 0.013 | 0.0620 | 0.234 |
| Avg state-space size | 0.761 | 16.671 | 222.4 |
| Max state-space size | 11.50 | 382.79 | 6,222 |
| Min state-space size | 0.007 | 0.0327 | 0.103 |

procedure for the J30 data set. Note that we use the SCV to represent the level of activity duration variability. From Table 6, it is clear that computational requirements increase with the number of phases. When compared to the procedure of Creemers (2015), however, we see that our approach can cope with much lower levels of variability, and is still able to solve all instances of the J30 data set if the activity duration distributions have 3 or 4 phases.

# 9 Conclusion

In this article, we tackled the PSRCPSP; an extension of the SRCPSP where activities are allowed to be interrupted. We are the first to study the PSRCPSP, and use a backward SDP recursion to determine the optimal makespan of a resource-constrained project that has preemptable activities and stochastic activity durations. We develop a new CTMC that, when compared to the well-known CTMC of Kulkarni and Adlakha (1986), drastically reduces memory requirements. In addition, we propose a new and efficient approach to structure the state space of the CTMC. These improvements allow us to easily outperform the current state-of-the-art in optimal project scheduling procedures. We are able to solve all instances of the PSPLIB J30 and J60 data sets with small computational effort. We also solve 196 (out of 480) instances of the PSPLIB J90 data set, and have even succeeded in solving some of the instances of the PSPLIB J120 data set. In addition, we show that our solutions are globally optimal. We also show that, if activity durations are exponentially distributed, the globally optimal policy for the SRCPSP and the PSRCPSP is an elementary policy (i.e., a policy that allows to make decisions at the end of activities, and at the start of the project).

Even though we focus on the PSRCPSP, the approach developed in this article is quite general, and can be applied to other scheduling problems as well. Our approach is especially suited for studying scheduling problems where the execution of an activity is allowed to be interrupted. If activities are non-preemptable, on the other hand, our approach can be used to determine lower bounds. In addition, our approach enumerates all subsets of potentially ongoing activities to determine the optimal set of ongoing activities. Heuristic procedures can be developed that allow to determine a "good" set of ongoing activities. Such heuristic procedures would significantly reduce the computational effort required to obtain a solution. Another direction for future research is to consider project scheduling problems where eNPV, rather than makespan, is considered as an objective.

# References

Ashtiani B, Leus R, Aryanezhad MB (2011) New competitive results for the stochastic resource-constrained project scheduling problem: Exploring the benefits of pre-processing. *J. Scheduling*

14(2):157-171.

Ballestín F (2007) When it is worthwhile to work with the stochastic RCPSP. *J. Scheduling* 10(3):153–166.

Ballestín F, Valls V, Quintanilla S (2008) Pre-emption in resource-constrained project scheduling. *Eur. J. Oper. Res.* 189(3):1136-1152.

Ballestín F, Valls V, Quintanilla S (2009) Scheduling projects with limited number of preemptions. *Comput. Oper. Res.* 36(11):2913-2925.

Ballestín F, Leus R (2009) Resource-constrained project scheduling for timely project completion with stochastic activity durations. *Prod. Oper. Management* 18(4):459–474.

Buss AH, Rosenblatt MJ (1997) Activity delay in stochastic project networks. *Oper. Res.* 45(1):126-139.

Creemers S, Leus R, Lambrecht M (2010) Scheduling Markovian PERT networks to maximize the net present value. *Oper. Res. Lett.* 38(1):51-56.

Creemers S, Leus R, De Reyck B (2015) Project planning with alternative technologies in uncertain environments. *Eur. J. Oper. Res.* 242(2):465–476.

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

Damay J, Quilliot A, Sanlaville E (2007) Linear programming based algorithms for preemptive and non-preemptive RCPSP. *Eur. J. Oper. Res.* 182(3):1012-1022.

Demeulemeester E, Herroelen W (1992) A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Sci.* 38(12):1803–1818.

Demeulemeester E, Herroelen W (1996) An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem. *Eur. J. Oper. Res.* 90(2):334-348.

Demeulemeester E, Herroelen W (1992) New Benchmark Results for the Resource-Constrained Project Scheduling Problem. *Management Sci.* 43(11):1485–1492.

Golenko-Ginzburg D, Gonik A (1997) Stochastic network project scheduling with non-consumable limited resources. *Internat. J. Production Econ.* 48(1):29–37.

Gutin E, Kuhn D, Wiesemann W (2015) Interdiction Games on Markovian PERT Networks. *Management Sci.* 61(5):999–1017.

Herroelen W, Leus R (2004) Robust and reactive project scheduling: Review and classification of procedures. *Internat. J. Production Res.* 42(8):1599–1620.

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

Kaplan L (1988) *Resource-constrained project scheduling with preemption of jobs*, PhD thesis. (University of Michigan).

Kolisch R, Sprecher A (1996) PSPLIB: A project scheduling problem library. *Eur. J. Oper. Res.* 96(1):205-216.

Kulkarni V, Adlakha V (1986) Markov and Markov-regenerative PERT networks. *Oper. Res.* 34(5):769-781.

Möhring RH (2000) Scheduling under uncertainty: Optimizing against a randomizing adversary. Jansen K, Khuller S, eds. *Lecture Notes in Computer Science Vol. 1913.* (Springer, New York), 15–26.

Moukrim A, Quilliot A, Toussaint H (2015) An effective branch-and-price algorithm for the Preemptive Resource Constrained Project Scheduling Problem based on minimal Interval Order Enumeration. *Eur. J. Oper. Res.* 244(2):360-368.

Neuts MF (1981) *Matrix-Geometric Solutions in Stochastic Models.* (Johns Hopkins University Press).

Osogami T (2005) *Analysis of multiserver systems via dimensionality reduction of Markov chains*, PhD thesis. (Carnegie Mellon University, Pittsburgh).

Patterson JH (1984) A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem. *Management Sci.* 30(7):854–867.

Van Peteghem V, Vanhoucke M (2010) A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. *Eur. J. Oper. Res.* 201(2):409-418.

Rostami S, Creemers S, Leus R (to appear) New strategies for stochastic resource-constrained project scheduling. *J. Scheduling.*

Shier DR, Whited DE (1986) Iterative algorithms for generating minimal cutsets in directed graphs. *Networks.* 16(2):133–147.

Sobel MJ, Szmerekovsky JG, Tilson V (2009) Scheduling projects with stochastic activity duration to maximize expected net present value. *Eur. J. Oper. Res.* 198(1):697-705.

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

Tsai Y-W, Gemmill DD (1998) Using tabu search to schedule activities of stochastic resource-constrained projects. *Eur. J. Oper. Res.* 111(1):129–141.

Vanhoucke M, Debels D (2008) The impact of various activity assumptions on the lead time and resource utilization of resource-constrained projects. *Comput. Ind. Eng.*, 54(1):140-154.

**FACULTY OF ECONOMICS AND BUSINESS**
Naamsestraat 69 bus 3500
3000 LEUVEN, BELGIË
tel. + 32 16 32 66 12
fax + 32 16 32 67 91
info@econ.kuleuven.be
www.econ.kuleuven.be