doi:10.1016/j.ejor.2019.02.030 • www.stefancreemers.be • info@stefancreemers.be

The preemptive stochastic resource-constrained project scheduling problem

Stefan Creemers

Abstract - **Preemption (or the splitting of activities) is a common practice in many project environments, and has been a standard feature of commercial project management software packages for years. Despite its prevalence in daily practice, preemption has received little attention in the project scheduling literature. A possible explanation for this lack of research interest is the common assumption that preemption only has a limited impact on the optimal makespan of a project. In this article, however, we show that the benefit of preemption can be significant, and that it increases with the size and the complexity of the project network. In addition, we also investigate how activity duration variability impacts the benefits of preemption. To this end, we study the preemptive stochastic resource-constrained project scheduling problem (PSRCPSP), and present an exact solution procedure. 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 hypoexponential distributions to model the 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) (Operations Research, 34(5), 769–781). In addition, we also propose a new and efficient approach to structure the state space of the CTMC.**

Keywords - **project scheduling, resource constraints, preemption, stochastic durations, continuous-time Markov chain**

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 PRCPSP was first investigated by Słowiński (1980),

who uses a large-scale linear program to solve several resource-constrained project scheduling problems. It is important to note that Słowiński (1980) considers continuous (or rational) preemption, where activities are allowed to be interrupted at arbitrary points in time. Integer preemption, on the other hand, assumes that activities can only be interrupted at integer points in time (see e.g., Kaplan, 1988; Demeulemeester and Herroelen, 1996). In this article, we focus on the case of continuous preemption. Even though Słowiński already published his seminal paper in 1980, the PRCPSP with continuous preemption was only picked up again in 2007, when Damay et al. developed a dedicated branch-and-bound algorithm that was used to solve all 480 instances of the PSPLIB J30 data set (Kolisch and Sprecher, 1996). They were unable, however, to obtain optimal solutions for the instances of the PSPLIB J60 data set. More recently, Moukrim et al. (2015) used a branch-and-price algorithm to solve all instances of the PSPLIB J30 data set, and to obtain lower bounds for instances of the PSPLIB J60, J90, and J120 data sets. An extensive review of the literature on the PRCPSP can be found in Schwindt and Paetz (2015), who also present a MILP formulation for the PRCPSP with continuous preemption.

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 exact 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, Ballestín and Leus (2009), Ashtiani et al. (2011), and Rostami et al. (2018). 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 start of the project and at the end of activities). 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. (2018) for a hierarchy of the different policy classes, and for an overview of the literature on the SRCPSP).

The preemptive SRCPSP (or PSRCPSP) is an extension of the SRCPSP that allows activities to be interrupted. In this article, we present an exact 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 hypoexponential distributions to model activity du-

rations, and match the first two moments of the "true" activity duration distribution (the hypoexponential distribution is a generalization of the Erlang distribution; a convolution of exponential distributions where each "phase" has its own exponential rate). To the best of our knowledge, we are the first to study the PSRCPSP. 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 10 instances of the PSPLIB J120 data set.

Most of the literature on stochastic project scheduling deals with Markovian PERT networks (i.e., PERT networks where the duration of the activities are exponentially distributed). 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 has 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, 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 state space. Although UDCs allow a strict partitioning of all feasible states, the generation of all UDCs 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 that also reduce the number of states that are stored in memory at any one time. These improvements allow us to easily outperform existing exact procedures that schedule Markovian PERT networks. Last but not least, we show that, if activity durations are exponentially distributed, elementary policies are dominant.

Not only is preemption common in practice (refer to Węglarz et al. (2011) and Schwindt and Paetz (2015) for an extensive list of applications), it is also a standard feature of most project management software packages. Even so, preemption has received only little attention in the project scheduling literature. A possible explanation for this lack of research interest is the common assumption that preemption only has a limited impact on the optimal makespan of a project (see Kaplan, 1988; Demeulemeester and Herroelen, 1996). For

instance, Demeulemeester and Herroelen (1996) report an average reduction of 0.78 % in the optimal makespan of the instances of the Patterson data set (Patterson, 1984). They conclude that "the introduction of preemption has little effect." This conclusion, however, is based solely on the analysis of the instances of the Patterson data set, and hence, may not hold in general. In addition, Kaplan, as well as Demeulemeester and Herroelen, consider integer (rather than continuous) preemption, and assume that activity durations are deterministic. If we allow for continuous preemption, on the other hand, there are more opportunities to benefit from preemption. To see this, one only needs to consider a project with three activities that have unit duration, and that each require a single unit of a resource that has an availability of two. The minimum project makespan of 1.5 time units can only be attained by suspending the execution of one activity at time 0.5, and resuming the same activity at time 1. In addition, if activity durations are stochastic, preemption might make more sense: activities with stochastic durations can take longer than expected, and may "lock down" a resource that is required to process another, more critical activity. By interrupting activities, such a "lockdown" may be resolved. To further investigate the impact of preemption, we perform an elaborate computational experiment.

Even though we focus on the PSRCPSP, the approach developed in this article is quite general, and can be used to tackle other scheduling problems as well (e.g., we have also applied it to maximize the expected net present value of a project without resource constraints, and to solve the precedence-constrained single machine weighted tardiness problem). Our approach is especially suitable 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 (see e.g., Brucker and Knust, 2003).

The contributions of this article can be summarized as follows: (1) we are the first to study the PSRCPSP, (2) we develop a new, exact procedure that outperforms existing state-of-the-art exact procedures that schedule Markovian PERT networks, (3) we present a new CTMC that drastically reduces memory requirements when compared to the well-known CTMC of Kulkarni and Adlakha (1986), (4) we present a new approach to structure the state space of the CTMC, and show that it significantly reduces memory requirements when compared to approaches that use UDCs to structure the state space, and (5) we perform an elaborate computational experiment to investigate the impact of preemption.

The remainder of this article is structured as follows. Section 2 presents basic definitions, and gives a brief problem statement. Section 3 defines the new CTMC, and explains how the hypoexponential distribution is used to model activity durations. Section 4 presents the backward SDP recursion, and outlines the approach that is used to structure the state space of the CTMC. Section 5 provides a numerical example, and Section 6 assesses the

computational performance of our procedure. The benefits of preemption are investigated in Section 7. Section 8 concludes.

2 Definitions and problem statement

A project is a network of activities that can be represented by a graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ 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 1 and n are dummy activities that correspond to the start and the completion of the project, respectively. The duration of an activity i is a random variable $\tilde{p}_i$ that has expected value μ_i, and that is independently distributed. $\tilde{\mathbf{p}} = \{\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_n\}$ denotes the vector of the activity duration random variables, and $\mathbf{p} = \{p_1, p_2, \ldots, p_n\}$ is a realization of $\tilde{\mathbf{p}}$, where p_i is a realization of $\tilde{p}_i$. 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 $r_{1,k} = r_{n,k} = 0$, for all $k \in \mathscr{R} = \{1, 2, \ldots, K\}$.

A solution to the deterministic RCPSP is a schedule $S = \{S_1, S_2, \ldots, S_n\}$, where S_i is the starting time of activity i, $S_1 = 0$, and S_n represents the completion time of the project. In addition, define $\mathscr{A}(S, t) = \{i \in V : S_i \leqslant t \wedge (S_i + p_i) > t)$, the set of activities in schedule S that are active at time t. A schedule S is feasible if:

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

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

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

An optimal schedule $S^\star$ minimizes S_n subject to Constraints (1–3).

The PRCPSP extends the RCPSP by allowing activities to be interrupted at arbitrary (continuous preemption) or integer (integer preemption) points in time. If an activity is interrupted, its execution can be seen as a series of non-consecutive stages. More formally, an activity i may be split into z_i stages that each require $r_{i,k}$ units of resource k, for all $k \in \mathscr{R}$. In addition, let $p_{z,i}$ denote the duration of stage z_i of activity i. A solution to the PRCPSP is a schedule $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_{2,1}, \ldots, \mathcal{S}_{2,z_2}, \mathcal{S}_{3,1}, \ldots, \mathcal{S}_n\}$, where $\mathcal{S}_{i,z}$ is the starting time of stage $z : 0 < z \leqslant z_i$ of activity i, $\mathcal{S}_1 = 0$, and $\mathcal{S}_n$ represents the completion time of the project. In addition, define, $\mathscr{A}(\mathcal{S}, t) = \{i \in V : \exists\ 0 < z \leqslant z_i$ for which $\mathcal{S}_{i,z} \leqslant t$ and $(\mathcal{S}_{i,z} + p_{z,i}) > t\}$,

the set of activities in schedule $\mathcal{S}$ that are active at time t. A schedule $\mathcal{S}$ is feasible if:

$$\mathcal{S}_{i,z} + p_{z,i} < \mathcal{S}_{i,(z+1)} \qquad \forall i \in V,\ \forall 0 < z < z_i, \tag{4}$$

$$\mathcal{S}_{i,z_i} + p_{z_i,i} \leqslant \mathcal{S}_{j,1} \qquad \forall (i,j) \in E, \tag{5}$$

$$\sum_{i \in \mathscr{A}(\mathcal{S},t)} r_{i,k} \leqslant R_k \qquad \forall t \geqslant 0,\ \forall k \in \mathscr{R}, \tag{6}$$

$$\mathcal{S}_{i,z} \geqslant 0 \qquad \forall i \in V,\ \forall 0 < z \leqslant z_i. \tag{7}$$

$$\sum_{z=1}^{z_i} p_{z,i} = p_i \qquad \forall i \in V, \tag{8}$$

$$p_{z,i} > 0 \qquad \forall i \in V,\ \forall 0 < z \leqslant z_i, \tag{9}$$

$$z_i \in \mathbb{Z}_{\geqslant 0} \qquad \forall i \in V, \tag{10}$$

An optimal schedule $\mathcal{S}^\star$ minimizes $\mathcal{S}_n$ subject to Constraints (4–10). If integer (rather than continuous) preemption is considered, Constraints (7) and (9) are replaced by:

$$\mathcal{S}_{i,z} \in \mathbb{Z}_{\geqslant 0} \qquad \forall i \in V,\ \forall 0 < z \leqslant z_i,$$

$$p_{z,i} \in \mathbb{Z}_{> 0} \qquad \forall i \in V,\ \forall 0 < z \leqslant z_i.$$

The PSRCPSP is an extension of the PRCPSP where activities 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 Π 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 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 Π may be interpreted as a function $\mathbb{R}^n_{\geqslant 0} \mapsto \mathbb{R}^n_{\geqslant 0}$ that maps realizations of activity durations $\mathbf{p}$ to vectors of feasible starting times $S(\mathbf{p};\Pi) = \{S(p_1;\Pi), S(p_2;\Pi), \ldots, S(p_n;\Pi)\}$ (see also Igelmund and Radermacher, 1983; Stork, 2001). For a given realization $\mathbf{p}$ and policy Π, $S_n(\mathbf{p};\Pi)$ 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 start of the project and at the end of activities. Even though elementary policies are not dominant if activity durations have arbitrary distributions (see e.g., Rostami et al., 2018), it is easy to show that there exists at least one dominant policy that is also

an elementary policy if activity durations are exponentially distributed. To see this, imagine that we want to make a decision (to start/interrupt a set of activities) at a time t that is not the start of the project nor the end of an activity. Due to the memoryless property of the exponential distribution, an ongoing activity i at time t has a remaining duration that is exponentially distributed with rate parameter λ_i (i.e., the expected remaining time for activity i does not change). As a result, at time t, no new information has become available that was not already available at time t', where t' is the start of the project or the last completion time of an activity. In other words, any decision taken at time t could already have been taken at time t', and therefore decisions taken at the start of the project/at the end of activities are dominant.

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 called 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 to develop scheduling procedures (see e.g., Buss and Rosenblatt, 1997; Sobel et al., 2009; Creemers et al., 2010; Creemers, 2015; Gutin et al., 2015). In the CTMC of Kulkarni and Adlakha, 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 selects 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 λ_i at any time instance t during which activity i is ongoing. 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 suitable for scheduling problems that allow the execution of activities to be interrupted. In addition, our CTMC is also suitable

for solving the preemptive-repeat problem, in which the work done on an activity is lost if the activity is interrupted (see e.g., Cai et al., 2009).

More formally, let $F(t)$ denote the set of 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) $i \notin F(t)$ and (2) $j \in F(t)$ for all j for which $(j,i) \in E$. The starting and finishing conditions of the project are $F(0) = \{1\}$ and $F(t) = V$ for all $t \geqslant \tau$, where τ 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 Π determines the non-empty set of ongoing activities $O \subseteq H$. Of course, O has to be a resource-feasible set of activities: $R_k \geqslant \sum_{i \in O} r_{i,k}$ for all $k \in \mathscr{R}$. An 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 Π 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 $i : i \in O$ finishes first equals $\lambda_i (\sum_{j \in O} \lambda_j)^{-1}$. Therefore, if policy Π is adopted, the time until completion of the project upon entry of state (F) equals:

$$G(\Pi, F) = \left(1 + \sum_{i \in O} \lambda_i G\left(\Pi, F \cup \{i\}\right)\right) \times \left(\sum_{i \in O} \lambda_i\right)^{-1}. \tag{11}$$

The optimal subset of ongoing activities is given by:

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

The problem of selecting $O^\star$ from H may be defined as a multidimensional $0-1$ knapsack problem:

$$\text{(MKP)} \begin{cases} \max \sum_{i \in H} x_i v_i. \\ \text{Subject to:} \\ \quad \sum_{i \in H} x_i r_{i,k} \leqslant R_k \quad \forall t \geqslant 0, \forall k \in \mathscr{R}, \\ \quad x_i \in \{0,1\} \qquad \forall i \in H, \end{cases}$$

where v_i is the value of selecting activity i, $x_i = 1$ if activity i is selected, and $x_i = 0$ otherwise. The value of selecting activity i depends on the selection that is made, and selecting activity i also impacts the value of other selected activities $j : j \in H \setminus \{i\} \wedge x_j = 1$. Even without such

a dependency, finding the optimal solution to (MKP) is NP-hard (Gens and Levner, 1980; Korte and Schrader, 1981). A brute-force approach that enumerates/evaluates all resource-feasible subsets of H is a straightforward means to obtain $O^\star$. In a preemptive context, however, non-delay schedules (see e.g., Sprecher et al., 1995) are dominant, and therefore we only need to consider maximal resource-feasible antichains of ongoing activities (i.e., it suffices to consider all precedence- and resource-feasible sets of ongoing activities that are not a subset of any other precedence- and resource-feasible set of ongoing activities).

Most of the literature on stochastic project scheduling deals with Markovian PERT networks. The assumption of exponentially-distributed activity durations, however, is not always valid, and therefore several researchers have resorted to the use of phase-type (PH) distributions to model activity durations (see e.g., Sobel et al., 2009; Creemers et al., 2010; Creemers, 2015; Creemers et al., 2015). PH distributions are a general class of distributions that use exponential distributions as building blocks (i.e., "phases"). Popular examples of PH distributions include the Erlang distribution, the hyperexponential distribution, and the exponential distribution. Because the phases of a PH distribution have exponentially-distributed durations, a project network with PH-distributed activity durations can be transformed into a Markovian PERT network (for further details, refer to Creemers, 2015). PH distributions are particularly useful because they can be used to match any positive-valued distribution with arbitrary precision (Neuts, 1981). In general, however, the number of phases (i.e., the complexity of the PH distribution) increases with the level of precision that is required. 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. Although it is possible to match more than two moments (see e.g., Altiok, 1985; Osogami and Harchol-Balter, 2006), matching only two moments: (1) limits the size of the resulting Markovian PERT network, and (2) makes practical sense because the "true" duration distribution of an activity is often unknown (i.e., in practice, we often only know the mean and the variance of the duration 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 σ_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 hypoexponential distribution is used to model the duration distribution. The hypoexponential distribution can be seen as a series of exponentially-distributed phases whose rate parameters are allowed to differ (i.e., the hypoexponential 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:

$$z_i = \lceil \nu_i^{-1} \rceil.$$

For the sake of simplicity, we assume that the first $z_i - 1$ phases of the hypoexponential 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; Rostami et al., 2018).

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; Gutin et al., 2015). Each state of the CTMC is assigned to a single UDC, and a UDC network is constructed (where a UDC is a predecessor of another UDC if it is possible to make a transition from one of its states to one of the states of the other UDC). UDCs are processed one at a time using a backward recursion. As soon as a UDC is no longer needed (i.e., if all its predecessor UDCs have been processed), the UDC and its associated states are removed from memory. Although UDCs allow a strict partitioning of the state space, generating all UDCs is an NP-hard problem (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 that 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 $(F) = \{1\}$. The minimum expected makespan equals $G(\Pi^\star, \{1\}) = E\left(S_n\left(\mathbf{p}; \Pi^\star\right)\right)$.

In addition, define $\mathbf{X}_i = \{(F) : |F| = i\}$, the array of states (F) for which i activities are finished, for all $i : 1 \leqslant i \leqslant n$. For each state (F), we keep track of value function $G(\Pi^\star, F)$.

In continuous time, at most one activity can finish at a given point in 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 determine 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 significant reduction of memory requirements when compared to scheduling procedures that rely on UDCs to structure the state space. In addition, our new approach is less impacted by the size of the network/is better suited for analyzing large projects (i.e., no matter the size of n, we only need to keep track of two arrays).

We use Eq. (11) to determine the value function of a state $(F) \in \mathbf{X}_i$, and use binary search to quickly look up 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 by enumerating all maximal resource-feasible antichains of ongoing activities. 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). States in $\mathbf{X}_{i+1}$ are ordered from small binary number to large binary number, and new states (in $\mathbf{X}_i$) are generated in the same order. More specifically, for each state (F) in $\mathbf{X}_{i+1}$, we generate a new state $(F \setminus \{j\}) \in \mathbf{X}_i$ for each activity $j : j \in F$ that has no successors in (F). Activities with a high index are used first in order to ensure that new states with small binary number are generated first. In addition, if the new state has a binary number that is smaller than/equal to the largest binary number of the already generated states, that new state already exists. Hence, in comparison with scheduling procedures that use UDCs to structure the state space, we significantly improve computational performance because: (1) we no longer have to search the set of already generated states to find out whether or not a newly generated state already exists, and (2) we no longer need to generate all UDCs, which in itself is an NP-hard problem.

After all value functions of all states (F) 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 $G(\Pi^\star, \{1\})$, the value function of state $(F) = \{1\}$, and have determined the minimum expected makespan of the project.

i	μ_i	$r_{i,1}$
1	0	0
2	3	3
3	2	2
4	2	5
5	1	2
6	0	0
R_1	5	

Table 1: Data for the example project

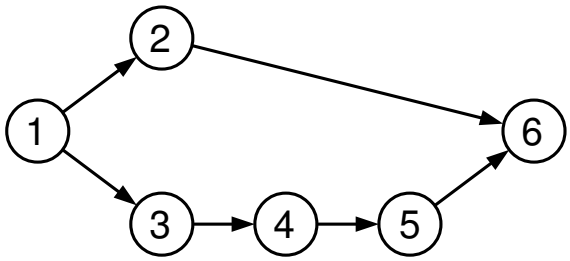

Figure 1: Example project network

5 Example

We use an example to illustrate the dynamics of our procedure as well as the difference between the RCPSP, the SRCPSP, the PRCPSP, and the PSRCPSP. The data of the example project is summarized in Table 1. Fig. 1 visualizes the example project network. 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 2 can be executed in parallel with activities 3, 4, and 5.

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. Fig. 2 and Fig. 3 illustrate the optimal deterministic schedule and the optimal stochastic policy, respectively. The optimal stochastic policy starts activities 2 and 3 at the start of the project. We expect either activity 2 (with probability 0.4) or activity 3 (with probability 0.6) to finish after 1.2 time units. If activity 2 finishes first, activities 3, 4, and 5 are executed in series. If activity 3 finished first, on the other hand, activities 2, 4, and 5 are executed in series.

Next, we observe what happens if the execution of an activity is allowed to be interrupted.

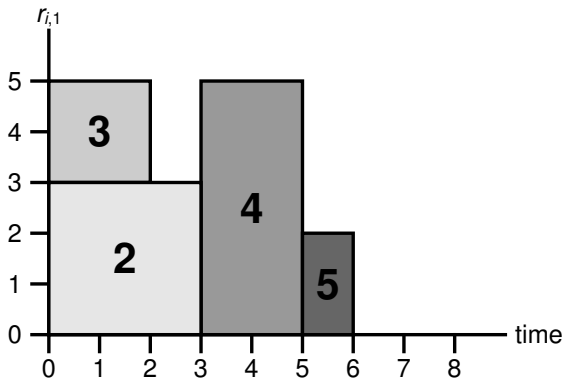

Figure 2: Optimal schedule if activities have deterministic durations and their execution cannot be interrupted

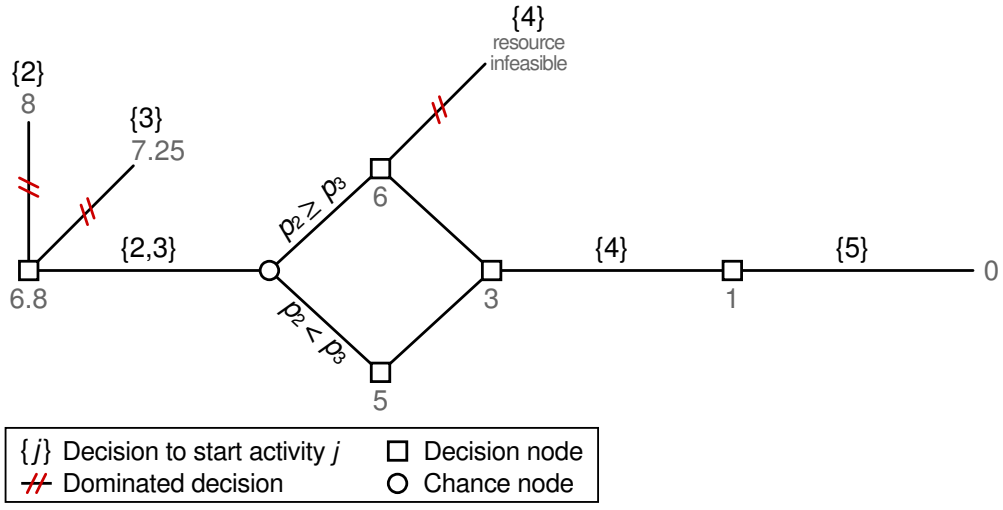

Figure 3: Optimal policy if activities have exponential durations and their execution cannot be interrupted

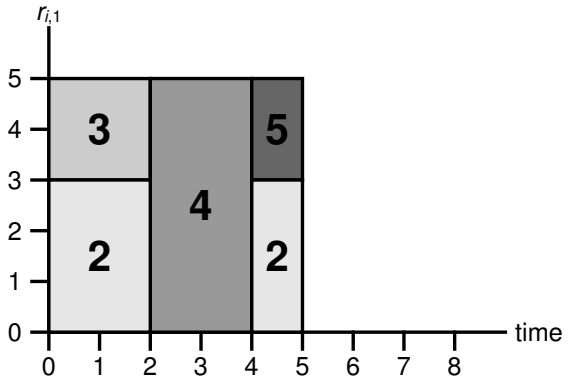

Figure 4: Optimal schedule if activities have deterministic durations and their execution can be interrupted

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. Fig. 4 and Fig. 5 illustrate the optimal deterministic schedule and the optimal stochastic policy, respectively. The optimal stochastic policy starts activities 2 and 3 at the start of the project. If activity 2 finishes first, activities 3, 4, and 5 are executed in series. If, on the other hand, activity 3 finishes first, the execution of activity 2 is interrupted, and activity 4 is started. After completion of activity 4, activity 2 is resumed and is executed in parallel with activity 5. In other words, the optimal policy tries to save time by executing activity 2 in parallel with activities 3 and/or 5.

We can also use the example project to illustrate how new states are generated/structured using our approach. First, array $\mathbf{X}_6 = \{(F_{6,1})\}$ is initialized, where $(F_{6,1}) = V = \{1, 2, 3, 4, 5, 6\}$. Only the dummy-end activity is without successors in $(F_{6,1})$, and therefore $\mathbf{X}_5 = \{(F_{5,1})\}$, with $(F_{5,1}) = \{1, 2, 3, 4, 5\}$. In $(F_{5,1})$ activities 2 and 5 are without successor. As a result, we first remove activity 5 (the activity with the highest index) to generate $(F_{4,1}) = \{1, 2, 3, 4\}$. Next, we remove activity 2, and generate $(F_{4,2}) = \{1, 3, 4, 5\}$. States $(F_{4,1})$ and $(F_{4,2})$ correspond to binary numbers 30 and 58, respectively, and they are ordered from small to large. To generate the states in $\mathbf{X}_3$, we first observe $(F_{4,1})$. In $(F_{4,1})$, activities 2 and 4 are without successor, and we generate $(F_{3,1}) = \{1, 2, 3\}$ and $(F_{3,2}) = \{1, 3, 4\}$ with corresponding binary numbers 14 and 26, respectively. Next, we observe $(F_{4,2})$, and see that only activity 5 is without successor. As such, we can generate state $(1, 3, 4)$ with binary number 26. Binary number 26, however, is smaller than/equal to the largest binary number of the states that were already generated (i.e., binary number 26 corresponding to state $(F_{3,2})$). As a result, state $(1, 3, 4)$ already exists, and all states in $\mathbf{X}_3$ have been identified. The same logic can be used to identify $\mathbf{X}_2 = \{(F_{2,1}), (F_{2,2})\}$, and $\mathbf{X}_1 = \{(F_{1,1})\}$, where $(F_{2,1}) = \{1, 2\}$,

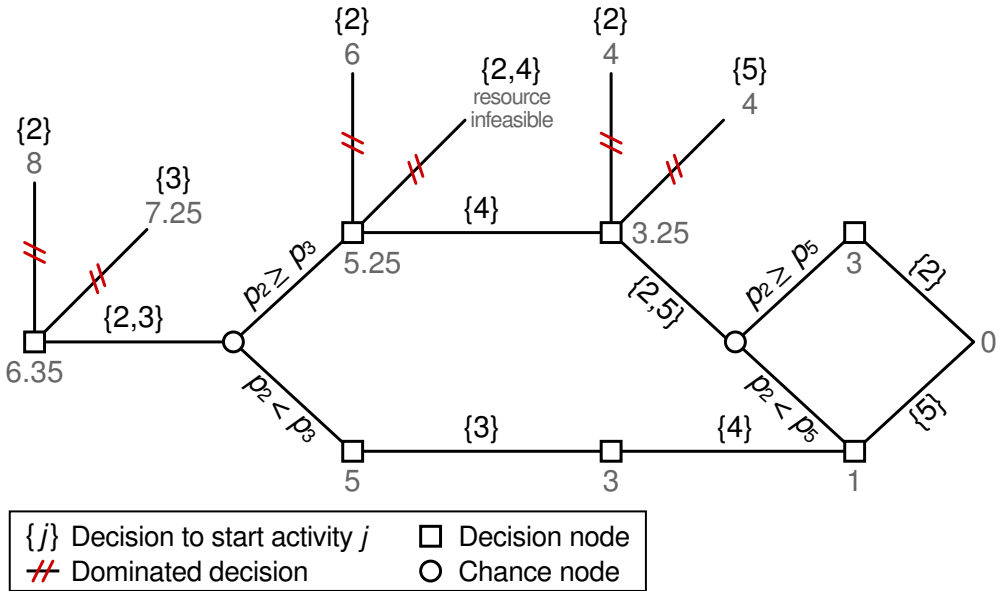

Figure 5: Optimal policy if activities have exponential durations and their execution can be interrupted

$(F_{2,2}) = \{1, 3\}$, and $(F_{1,1}) = \{1\}$.

In our approach, the CTMC has 9 states that are listed in Table 2. For each state, Table 2 also reports the optimal set of ongoing activities and the corresponding value function. For instance, in state $(F_{2,2})$ it is optimal to start activity 4 (no matter whether or not activity 2 is ongoing; if activity 2 was ongoing, it should be interrupted). If, on the other hand, the CTMC of Kulkarni and Adlakha (1986) is used, the state space can be structured in 5 UDCs: $\text{UDC}_1 = \{1\}$, $\text{UDC}_2 = \{2, 3\}$, $\text{UDC}_3 = \{2, 4\}$, $\text{UDC}_4 = \{2, 5\}$, and $\text{UDC}_5 = \{6\}$. The UDCs and their states are listed in Table 3, that also reports the optimal set of ongoing activities and the corresponding value function. For instance, in UDC_3, we observe activities 2 and 4 (i.e., we assume that activities 1 and 3 are finished, and that activities 5 and 6 are idle/not yet available). We end up in UDC_3 if activity 3 finishes in any of the states of UDC_2. We leave UDC_3 as soon as activity 3 completes. From Table 3, we can see that, in UDC_3, it is optimal to start activity 4 no matter whether activity 2 is idle, ongoing, or finished. In total, the CTMC of Kulkarni and Adlakha has 24 states of which at most 13 states need to be kept in memory at any one time (versus 4 states in our approach).

Array	State	F	$O^\star$	$G(\Pi^\star, F)$
$\mathbf{X}_6$	$F_{6,1}$	$\{1,2,3,4,5,6\}$	$\emptyset$	0
$\mathbf{X}_5$	$F_{5,1}$	$\{1,2,3,4,5\}$	$\{6\}$	0
$\mathbf{X}_4$	$F_{4,1}$	$\{1,2,3,4\}$	$\{5\}$	1
$\mathbf{X}_4$	$F_{4,2}$	$\{1,3,4,5\}$	$\{2\}$	3
$\mathbf{X}_3$	$F_{3,1}$	$\{1,2,3\}$	$\{4\}$	3
$\mathbf{X}_3$	$F_{3,2}$	$\{1,3,4\}$	$\{2,5\}$	3.25
$\mathbf{X}_2$	$F_{2,1}$	$\{1,2\}$	$\{3\}$	5
$\mathbf{X}_2$	$F_{2,2}$	$\{1,3\}$	$\{4\}$	5.25
$\mathbf{X}_1$	$F_{1,1}$	$\{1\}$	$\{2,3\}$	6.35

Table 2: State space and optimal policy of the example project if we use our new CTMC and our new approach to structure the state space

6 Computational performance

Although we are the first to study the PSRCPSP, we can compare the computational performance of our approach with the performance of the procedures of Creemers (2015) and Moukrim et al. (2015): the current state-of-the-art procedures for solving the SRCPSP and the PRCPSP, respectively. In what follows, we first compare the performance of our procedure and the procedure of Creemers (2015). In contrast to the procedure of Creemers (2015), our procedure: (1) allows for preemption, (2) does not use the CTMC of Kulkarni and Adlakha (1986), (3) does not rely on UDCs to structure the state space, and (4) can exploit the fact that non-delay schedules are dominant if preemption is allowed. As explained in Section 3, if non-delay schedules are dominant, the optimal set of ongoing activities can be found by enumerating all maximal resource-feasible antichains (rather than using a "brute-force" approach that enumerates all feasible sets of ongoing activities). If preemption is not allowed, however, non-delay schedules are not necessarily optimal, and hence, the procedure of Creemers (2015) cannot exploit this dominance rule. In order to make a fair comparison, we therefore first assess the performance of our procedure without exploiting the dominance rule (i.e., we enumerate all feasible sets of ongoing activities; we use a "brute-force" approach). The impact of the dominance property itself is assessed afterwards, when we compare the performance of our procedure and the procedure of Moukrim et al. (2015). To allow for a fair comparison, we perform all tests on the same system: an Intel I5 3.3 GHz personal computer with 32 GB of RAM.

Table 4 reports on the difference in performance between our procedure and the procedure

UDC	Members	F	O	I	$O^\star$	$G(\Pi^\star, F, O, I)$
UDC_5	$\{6\}$	$\{6\}$	$\emptyset$	$\emptyset$	$\emptyset$	0
		$\emptyset$	$\{6\}$	$\emptyset$	$\{6\}$	0
		$\emptyset$	$\emptyset$	$\{6\}$	$\{6\}$	0
UDC_4	$\{2,5\}$	$\{5\}$	$\{2\}$	$\emptyset$	$\{2\}$	3
		$\{5\}$	$\emptyset$	$\{2\}$	$\{2\}$	3
		$\{2\}$	$\{5\}$	$\emptyset$	$\{5\}$	1
		$\{2\}$	$\emptyset$	$\{5\}$	$\{5\}$	1
		$\emptyset$	$\{2,5\}$	$\emptyset$	$\{2,5\}$	3.25
		$\emptyset$	$\{5\}$	$\{2\}$	$\{2,5\}$	3.25
		$\emptyset$	$\{2\}$	$\{5\}$	$\{2,5\}$	3.25
		$\emptyset$	$\emptyset$	$\{2,5\}$	$\{2,5\}$	3.25
UDC_3	$\{2,4\}$	$\{2\}$	$\{4\}$	$\emptyset$	$\{4\}$	3
		$\{2\}$	$\emptyset$	$\{4\}$	$\{4\}$	3
		$\emptyset$	$\{4\}$	$\{2\}$	$\{4\}$	5.25
		$\emptyset$	$\{2\}$	$\{4\}$	$\{4\}$	5.25
		$\emptyset$	$\emptyset$	$\{2,4\}$	$\{4\}$	5.25
UDC_2	$\{2,3\}$	$\{2\}$	$\{3\}$	$\emptyset$	$\{3\}$	5
		$\{2\}$	$\emptyset$	$\{3\}$	$\{3\}$	5
		$\emptyset$	$\{2,3\}$	$\emptyset$	$\{2,3\}$	6.35
		$\emptyset$	$\{3\}$	$\{2\}$	$\{2,3\}$	6.35
		$\emptyset$	$\{2\}$	$\{3\}$	$\{2,3\}$	6.35
		$\emptyset$	$\emptyset$	$\{2,3\}$	$\{2,3\}$	6.35
UDC_1	$\{1\}$	$\emptyset$	$\{1\}$	$\emptyset$	$\{1\}$	6.35
		$\emptyset$	$\emptyset$	$\{1\}$	$\{1\}$	6.35

Table 3: State space and optimal policy of the example project if we use the CTMC of Kulkarni and Adlakha (1986) and UDCs to structure the state space

Data set	J30		J60	
Approach	SRCPSP	PSRCPSP	SRCPSP	PSRCPSP
Instances in set	480		480	
Instances solved/compared	480		303	
# activities	32		62	
Avg CPU time (s)	0.485	0.025	1,592	78.87
Max CPU time (s)	14.02	0.393	31,838	1,061
Min CPU time (s)	0.000	0.002	1.903	0.203
CPU improvement factor	19.52		20.18	
Avg state-space size	0.539	0.010	822.7	4.002
Max state-space size	11.38	0.072	4,257	32.85
Min state-space size	0.006	0.001	3.762	0.082
Memory improvement factor	53.60		205.6	
Avg max % in memory	28.60	17.11	42.04	11.18

Table 4: Comparison of computational performance with the procedure of Creemers (2015) if activity durations are exponentially distributed (state-space sizes are expressed in millions of states)

of Creemers (2015) for the instances of the PSPLIB data set. Note that, due to insufficient memory, the procedure of Creemers (2015) is unable to solve 177 (out of 480) instances of the J60 data set. Therefore, we can only compare the 303 instances that could be solved. From Table 4, however, it is clear that our procedure significantly outperforms the 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 (i.e., on average, our CTMC has 205 times less states than the CTMC of Kulkarni and Adlakha). In addition, Table 4 also reports the average maximum percentage of states that are stored in memory at any one time (i.e., the maximum proportion of the state space kept in memory, averaged over all instances). In contrast to the procedure of Creemers (2015), we no longer use UDCs, and as a result, the maximum percentage of states that have to be stored in memory has dropped significantly. If our new approach is used to structure the state space, we also observe that an increase in n results in a decrease of the percentage of states that are stored in memory.

The aforementioned results assume that activity durations are exponentially distributed. If, on the other hand, activity durations have a SCV smaller than 1, a hypoexponential distribution is used to model the duration distributions. Table 5 reports the computational performance of our procedure for the J30 data set for different levels of SCV. From Table 5, 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

Activity duration SCV	1/2	1/3	1/4
Number of phases	2	3	4
# 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

Table 5: 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)

lower levels of variability, and is still able to solve all instances of the J30 data set, even if the activity duration distributions have 3 or 4 phases.

We conclude that our procedure significantly outperforms the procedure of Creemers (2015), even if a brute-force approach is used to determine the optimal set of ongoing activities. If preemption is allowed, however, non-delay schedules are optimal, and we only need to consider maximal resource-feasible antichains of ongoing activities. Table 6 reports on the performance of our procedure depending on whether or not a brute-force approach was used to determine the optimal set of ongoing activities. From Table 6 it is clear that the dominance rule has a significant impact on the performance of the procedure. On average, computation times have reduced by a factor of 7.6, and the number of sets of ongoing activities that were evaluated has reduced by a factor of 4.6. Note, however, that there is no impact on the size of the state space. Table 6 and Fig. 6 also show that we can solve networks of up to 62 activities with small computational effort. We are able to solve 196 instances of the J90 data set, and have solved 10 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. Whereas memory is the main bottleneck of procedures that use the CTMC of Kulkarni and Adlakha (1986), the bottleneck of our approach is CPU time: an average of 17.6 hours to solve one instance of the J90 data set can hardly be called practical (note that no CPU time limit was imposed).

The procedure of Moukrim et al. (2015) is the current state-of-the-art for solving the PRCPSP. Moukrim et al. (2015) solve all instances of PSPLIB J30 (with an average CPU requirement of 1.75 seconds), and report solutions for 383 (out of 480) instances of the PSPLIB J60 data set when a time limit of 3 hours is imposed. For J90 and J120, they are

Data set	J30		J60		J90	
Brute-force approach	Yes	No	Yes	No	Yes	No
Instances in set	480		480		480	
Instances solved	480		480		196	
# activities	32		62		92	
Avg CPU time (s)	0.025	0.013	8.630e3	2.586e3	178.8e3	63,359
Max CPU time (s)	0.393	0.087	251.9e3	88.32e3	3,947e3	0.868e3
Min CPU time (s)	0.002	0.002	0.203	0.118	242.8	99.90
Improvement factor	1.934		3.338		8.022	
Avg state-space size	0.010	0.010	71.69	71.69	516.5	516.5
Max state-space size	0.072	0.072	962.2	962.2	4,245	4,245
Min state-space size	0.001	0.001	0.082	0.082	21.68	21.68
Improvement factor	NA		NA		NA	
Avg # sets of ongoing activities	0.521e6	0.117e6	0.194e12	0.036e12	3.779e12	0.845e12
Max # sets of ongoing activities	11.31e6	1.245e6	5.708e12	1.276e12	86.63e12	12.10e12
Min # sets of ongoing activities	0.005e6	0.001e6	3.680e6	0.618e6	3.555e9	837.7e6
Improvement factor	4.459		5.371		4.470	
Avg max % in memory	17.11		12.55		12.81	

Table 6: Comparison of computational performance depending on the approach that was used to find the optimal set of ongoing activities (state-space sizes are expressed in millions of states)

able to solve 299 (out of 480) and 21 (out of 600) instances within said time limit. They use both CPLEX and SCIP, and run their tests on an Intel Xeon 2.4 GHz computer. We conclude that our procedure outperforms the procedure of Moukrim et al. (2015) for small-to medium-sized projects (our procedure takes 0.013 seconds for solving an instance of the J30 data set, and can solve up to 449 instances of the J60 data set when a time limit of 3 hours is imposed). For larger projects, however, the procedure of Moukrim et al. (2015) still has the edge.

7 Benefit of preemption

To assess the benefit of preemption, we perform an elaborate experiment. We use RanGen (Demeulemeester et al., 2003) to generate 30 projects for each combination of:

- Project size ($n \in \{12, 22, 32, 42, 52, 62\}$).
- Order strength (OS $\in \{0.4, 0.6, 0.8\}$).

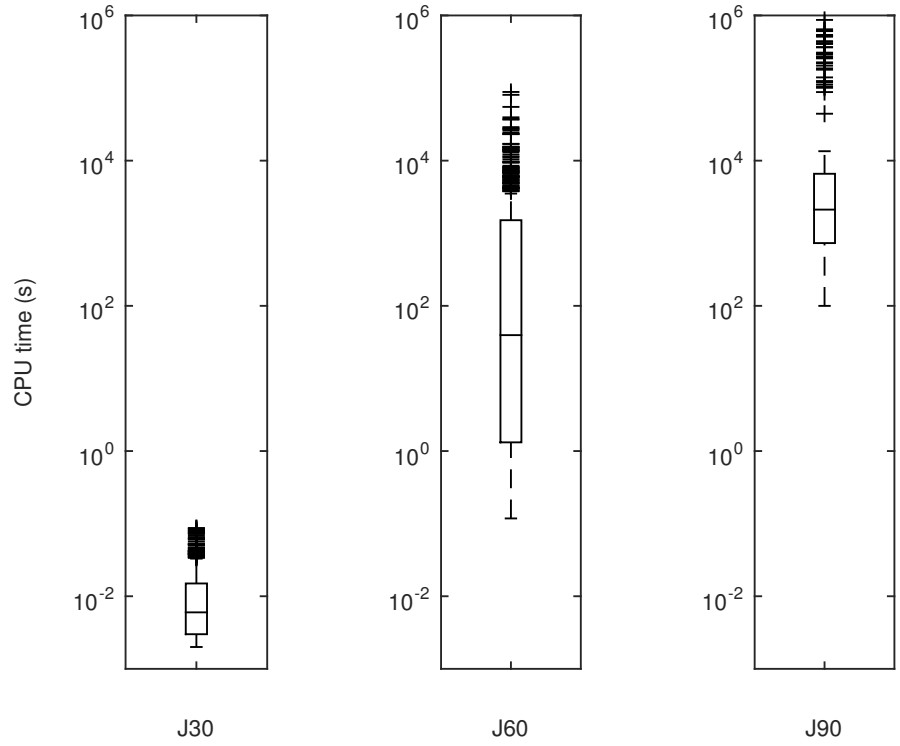

Figure 6: Computational performance on the PSPLIB data set if activity durations are exponentially distributed

- Resource constrainedness (RC $\in \{0.25, 0.5, 0.75\}$).

In addition, we consider three settings:

- No variability in activity durations (we compare the RCPSP and the PSRCPSP).
- Average variability in activity durations (we compare the SRCPSP and the PSRCPSP when activity durations follow a hypoexponential distribution with a SCV of 0.5).
- High variability in activity durations (we compare the SRCPSP and the PSRCPSP when activity durations are exponentially distributed).

For solving the RCPSP, we implemented the branch-and-bound algorithm of Demeulemeester and Herroelen (1997). For solving the PRCPSP with continuous preemption, we use the model of Moukrim et al. (2015), who provided us with a copy of their code. For solving the SRCPSP, we use the procedure of Creemers (2015).

Table 7 summarizes the results of the experiment. For each setting and for each combination of parameters, Table 7 reports the average gap in optimal makespan when preemption is allowed. The number between brackets next to the percentage gap indicates the number of instances (out of 30) that could not be solved. In contrast to popular belief, Table 7 shows that the benefit of preemption can be significant, regardless of the variability of the activity durations. We report average gaps of up to 5.46 %, and observe that the benefit of preemption increases with the complexity of the network (i.e., the order strength; a measure of the density of the project network) and the constrainedness of the resources (i.e., the ratio of the average requirement of a resource over its availability). In general, preemption was beneficial for 486, 793, and 776 instances (out of 1,437 instances that could be solved by all models) if the activity durations were deterministic, had a SCV of 0.5, and were exponentially distributed, respectively. On average, the benefit is largest if activity durations are deterministic (1.04 % gap) or had a SCV of 0.5 (0.93 % gap). If activity durations are exponentially distributed, the gap was only 0.65 % on average. As such, preemption is more often beneficial if activity durations are variable, however, the benefit itself is less outspoken.

8 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 expected makespan of a resource-constrained project that has preemptable activities and stochastic activity durations. We develop a new CTMC that,

SCV = 0 (RCPSP versus PRCPSP)									
OS		0.4			0.6			0.8	
RC	0.25	0.50	0.75	0.25	0.50	0.75	0.25	0.50	0.75
$n = 12$	3.55	0	0	0.42	0	0	0.41	0	0
$n = 22$	3.91	0.24	0	2.15	0.16	0	0.96	0.11	0
$n = 32$	4.78	0.48	0	2.84	0.53	0	1.71	0.31	0
$n = 42$	4.07	0.46	0	4.29	0.53	0	2.53	0.26	0
$n = 52$	NA	0.19	0	3.89	0.23	0	3.22	0.14	0
$n = 62$	NA	0.40	0	$3.87^{(2)}$	0.35	0	3.34	0.30	0
SCV = 0.5 (SRCPSP versus PSRCPSP)									
OS		0.4			0.6			0.8	
RC	0.25	0.50	0.75	0.25	0.50	0.75	0.25	0.50	0.75
$n = 12$	1.08	0.01	0	0.21	0.02	0	0.08	0	0
$n = 22$	2.63	0.38	0	1.21	0.20	0	0.23	0.21	0
$n = 32$	4.29	0.68	0	2.35	0.69	0	0.92	0.24	0
$n = 42$	$5.46^{(1)}$	0.95	0.01	3.71	0.82	0	1.13	0.42	0
$n = 52$	NA	NA	NA	4.67	0.70	0	1.84	0.47	0
$n = 62$	NA	NA	NA	5.23	1.07	0	2.63	0.58	0
SCV = 1 (SRCPSP versus PSRCPSP)									
OS		0.4			0.6			0.8	
RC	0.25	0.50	0.75	0.25	0.50	0.75	0.25	0.50	0.75
$n = 12$	0.48	0.02	0	0.11	0.04	0	0.01	0	0
$n = 22$	1.36	0.4	0	0.66	0.24	0	0.11	0.21	0
$n = 32$	2.57	0.75	0	1.38	0.73	0	0.49	0.26	0
$n = 42$	3.76	1.03	0.01	2.24	0.92	0	0.61	0.47	0
$n = 52$	4.57	1.18	0	3.15	0.8	0	1.04	0.55	0
$n = 62$	$5.15^{(3)}$	1.36	0	3.63	1.23	0	1.57	0.64	0

Table 7: Percentage gap in optimal makespan if preemption is allowed (the number of missing observations is indicated between brackets)

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 state-of-the-art exact procedures that schedule Markovian PERT networks (i.e., PERT networks where the duration of the activities are exponentially distributed). 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 10 instances of the PSPLIB J120 data set. In addition, we show that our solutions are optimal.

Even though preemption (or the splitting of activities) has many applications in real-life project environments, it has received only little attention in the project scheduling literature. A possible explanation for this lack of research interest is the common assumption that preemption only has limited impact on the optimal makespan of a project. To further investigate this claim, we perform an elaborate computational experiment. In contrast to popular belief, the experiment has shown that the benefit of preemption can be significant (up to 5.46% on average), especially for projects that have a complex network structure, and whose resources are heavily constrained. The experiment also reveals that preemption can be beneficial regardless the variability of the activity durations. In fact, on average, the benefit of preemption is largest when activity durations are deterministic.

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 suitable 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.

References

Altiok, T. (1985). On the phase-type approximations of general distributions. *IIE Transactions*, 17(2), 110–116.

Ashtiani, B., Leus, R., & Aryanezhad, M.B. (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., & Leus, R. (2009). Resource-constrained project scheduling for timely project completion with stochastic activity durations. *Production and Operations Management*, 18(4), 459–474.

Brucker, P., & Knust, S. (2003). Lower bounds for resource-constrained project scheduling problems. *European Journal of Operational Research*, 149(2), 302–313.

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

Cai, X., Wu, X., & Zhou, X. (2009). Stochastic scheduling subject to preemptive-repeat breakdowns with incomplete information. *Operations Research*, 57(5), 1236–1249.

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

Creemers, S., Leus, R., & De Reyck, B. (2015). Project planning with alternative technologies in uncertain environments. *European Journal of Operational Research*, 242(2), 465–476.

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

Damay, J., Quilliot, A., & Sanlaville, E. (2007). Linear programming based algorithms for preemptive and non-preemptive RCPSP. *European Journal of Operational Research*, 182(3), 1012–1022.

Demeulemeester, E., & Herroelen, W. (1996). An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem. *European Journal of Operational Research*, 90(2), 334–348.

Demeulemeester, E., & Herroelen, W. (1997). New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43(11), 1485–1492.

Demeulemeester, E., Vanhoucke, M., & Herroelen, W. (2003). RanGen: A random network generator for activity-on-the-node networks. *Journal of Scheduling*, 6(1), 17–38.

Gutin, E., Kuhn, D., & Wiesemann, W. (2015). Interdiction games on Markovian PERT networks. *Management Science*, 61(5), 999–1017.

Gens, G., & Levner, E. (1980). Complexity of approximation algorithms for combinatorial problems: A survey. *ACM SIGACT News*, 12(3), 52–65.

Herroelen, W., & Leus, R. (2004). Robust and reactive project scheduling: Review and classification of procedures. *International Journal of Production Research*, 42(8), 1599–1620.

Igelmund, G., & Radermacher, F.J. (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. *European Journal of Operational Research*, 96(1), 205–216.

Korte, B., & Schrader, R. (2000). On the existence of fast approximation schemes. In O.L. Mangasarian, R.R. Meyer, & S.M. Robinson (Eds.), *Nonlinear programming 4* (pp. 415–437). New York: Academic Press.

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

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. *European Journal of Operational Research*, 244(2), 360–368.

Neuts, M.F. (1981). *Matrix-geometric solutions in stochastic models: An algorithmic approach.* (The Johns Hopkins University Press, Baltimore).

Osogami, T., & Harchol-Balter, M. (2006). Closed form solutions for mapping general distributions to quasi-minimal PH distributions. *Performance Evaluation*, 63(6), 524–552.

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

Rostami, S., Creemers, S., & Leus, R. (2018). New strategies for stochastic resource-constrained project scheduling. *Journal of Scheduling*, 21(3), 349–365.

Schwindt, C., & Paetz, T. (2015). Continuous preemption problems. In C. Schwindt, & J. Zimmermann (Eds.), *Handbook on project management and scheduling* (pp. 251–295). Berlin: Springer.

Shier, D.R., & Whited, D.E. (1986). Iterative algorithms for generating minimal cutsets in directed graphs. *Networks*, 16(2), 133–147.

Słowiński, R. (1980). Two approaches to problems of resource allocation among project activities: A comparative study. *The Journal of the Operational Research Society*, 31(8), 711-723.

Sobel, M.J., Szmerekovsky, J.G., & Tilson, V. (2009). Scheduling projects with stochastic activity duration to maximize expected net present value. *European Journal of Operational Research*, 198(1), 697–705.

Sprecher, A., Kolisch, R., & Drexl, A. (1995). Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 80(1), 94–102.

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

Węglarz, J., Józefowska, J., Mika, M., & Waligóra, G. (2011). Project scheduling with finite or infinite number of activity processing modes A survey. *European Journal of Operational Research*, 208(3), 177–205.