KU LEUVEN

FACULTY OF ECONOMICS AND BUSINESS

Scheduling Markovian PERT networks to maximize the net present value: new results

Hermans B, Leus R.

KBI_1709

Scheduling Markovian PERT networks to maximize the net present value: New results

Ben Hermans*,[a] and Roel Leus†,[a]

[a] *Research Center for Operations Research & Business Statistics (ORSTAT), Faculty of Economics and Business, KU Leuven, Belgium*

Abstract

We study the problem of scheduling a project so as to maximize its expected net present value when task durations are exponentially distributed. Based on the structural properties of an optimal solution we show that, even if preemption is allowed, it is not necessary to do so. Next to its managerial importance, this result also allows for a new algorithm which improves on the current state of the art with several orders of magnitude, both in CPU time and in memory usage.

Keywords: project scheduling, net present value, exponentially distributed activity durations, Markov decision process, monotone optimal policy

1 Introduction

Consider a project in which a set of tasks $N = \{1, \ldots, n\}$ needs to be performed in order to reach a given goal; think for example of a new product development project. The strict partial order $A \subset N \times N$ defines precedence constraints, with $(i, j) \in A$ indicating that task j can only start if i is finished. We suppose task n reflects the project's completion and $(i, n) \in A$ for all $i \in N \setminus \{n\}$. Each task $i \in N$ has a random duration $\tilde{d}_i$ with support $\mathbb{R}_{\geq 0}$ and, initially, we require activities to be processed without interruption. Finally, each task has a cash flow which is discounted to take into account the time value of money.

In this article, we study the problem of deciding when to start each task so as to maximise the project's *expected net present value* (eNPV), i.e. the expected sum of discounted cash flows. Throughout, we assume independent and exponentially distributed activity durations $(\tilde{d}_i)_{i \in N}$ with rate parameter $(\lambda_i)_{i \in N} \in \mathbb{R}^n_{>0}$. This problem was also studied by [1, 2, 8]; we briefly review these articles below. For an excellent and more detailed literature review, also for the case where task durations are not exponentially distributed, see [10].

All models in [1, 2, 8] are based on the seminal work of Kulkarni and Adlakha [5], who use a continuous-time Markov chain to evaluate the moments and distribution of the project's earliest completion time. Buss and Rosenblatt [1]

*The research was funded by a PhD Fellowship of the Research Foundation – Flanders.

†Corresponding author. E-mail: roel.leus@kuleuven.be.

adapt this Markov chain to evaluate the eNPV when each task is initiated as soon as possible. Next, they determine the optimal delay for up to two activities beyond their earliest possible starting time. While the delays in [1] are fixed before the project's start, Sobel et al. [8] consider the more general case of making scheduling decisions adaptively during the project's execution, solving the problem using a stochastic dynamic program (SDP). To mitigate the SDP's excessive memory usage, Creemers et al. [2] partition the state space such that not all states have to be stored in memory at the same time. This significantly improves performance and their algorithm is considered to be the current state of the art [10].

We address the same problem as [2], which we define in Section 2 and for which we give a new SDP-formulation in Section 3. The major difference is that we act *as if* activities can be interrupted, but show that, even if preemption is allowed, it is not necessary to do so (Section 4). Consequently, the preemptive case solves the non-preemptive case as well. Next to its managerial importance, this result also allows for a new algorithm that improves on the method of [2] with several orders of magnitude, both in CPU time and in memory usage (Section 5).

2 Problem definition

We assume that executing task $i \in N$ leads to a cash flow $c_i \in \mathbb{R}$ per time unit, such that $c_i\tilde{d}_i$ equals the activity's total cash flow. Applying a continuous discount rate $r \in \mathbb{R}_{>0}$ to task i starting at time τ then yields a net present value $\int_\tau^{\tau+\tilde{d}_i} c_i e^{-rt}\,\mathrm{d}t$. We make the, often realistic, assumption that $c_i \leq 0$ for $i \in N \setminus \{n\}$ and $c_n > 0$; think e.g. of a new product development project where the company only obtains a revenue after launching the product [1].

A solution is a *policy* that specifies which task to start at which decision moment, possibly depending on events predating this decision moment. Denote the set of all feasible policies by Π, then any policy $\pi \in \Pi$ maps realisations $\boldsymbol{d} := (d_i)_{i\in N}$ of the random durations $\tilde{\boldsymbol{d}} := (\tilde{d}_i)_{i\in N}$ into starting times $(s_i(\boldsymbol{d} \mid \pi))_{i\in N}$. To be feasible, policy π has to respect the precedence constraints: $s_i(\boldsymbol{d} \mid \pi) + d_i \leq s_j(\boldsymbol{d} \mid \pi)$ for any $(i,j) \in A$ and any realisation $\boldsymbol{d}$ of $\tilde{\boldsymbol{d}}$. Moreover, the non-anticipativity constraint states that, when making a decision, the project manager cannot use information that becomes available after the decision moment. Formally, this means that, for any two realisations $\boldsymbol{d}$ and $\boldsymbol{d}'$ of $\tilde{\boldsymbol{d}}$, we have $s_i(\boldsymbol{d} \mid \pi) = s_i(\boldsymbol{d}' \mid \pi)$ as long as $d_j = d'_j$ for any $j \in N$ with $s_j(\boldsymbol{d} \mid \pi) + d_j \leq s_i(\boldsymbol{d} \mid \pi)$. We obtain the following problem:

$$\max_{\pi\in\Pi} \mathbb{E}\left[\sum_{i\in N} \int_{s_i(\tilde{\boldsymbol{d}}|\pi)}^{s_i(\tilde{\boldsymbol{d}}|\pi)+\tilde{d}_i} c_i e^{-rt}\,\mathrm{d}t\right].$$

Note that [2] assume *fixed* cash flows c_i^F occurring at task i's initiation. Such cash flows can be incorporated into our setting by choosing c_i such that $c_i^F = \mathbb{E}[\int_0^{\tilde{d}_i} c_i e^{-rt}\,\mathrm{d}t]$. Indeed, as can be derived from the expression above, this leads to the same decisions and eNPV, see [8, 10]. We model the cash flows as rates because this feels more natural for the preemptive case.

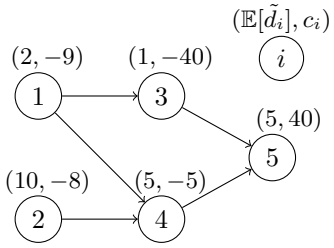

Figure 1: Example project.

Example. Figure 1 visualises an example project consisting of five activities $N = \{1, \ldots, 5\}$ represented on the nodes and precedence constraints as implied by the arrows. The arc $1 \to 3$, for instance, indicates that task 3 can only start after activity 1's completion. The expected task durations $\mathbb{E}[\tilde{d}_i] = 1/\lambda_i$ and cash flow rates c_i are shown above the nodes. Together with a continuous discount rate $r = 0.01$, this information fully specifies a problem instance.

One possible solution is the *early-start* policy π^e, which initiates each task as soon as possible. Using the procedure of [1], we find that the eNPV for π^e equals 13.49, while optimally delaying task 3 by 3.57 time units leads to an eNPV of 14.07. Using the SDP of [8], finally, we find that the maximum eNPV equals 16.59; the optimal policy $\pi^\star$ starts each activity different from task 3 as soon as possible and, for task 3, waits until tasks 1, 2 and 4 are past.

3 Stochastic dynamic program

In this section, we formulate a new stochastic dynamic program which assumes that we can interrupt a task at any time during its execution. In Section 4, we show that an optimal policy for this preemptive case solves the non-preemptive case as well and, thus, our SDP effectively gives a formulation for the problem of Section 2.

We define the project's *state* at any given time by the set $P \subseteq N$ of activities that are already finished (past). The project's *state space* $\mathcal{P}$ collects all states $P \subseteq N$ such that $i \in P$ implies $j \in P$ for all $(j, i) \in A$. The terminal state $P = N$ represents a completed project. The major difference compared to the SDPs of [2, 8] is that our state definition does not include the set of active tasks, i.e. tasks that are in process. Since preemption is allowed, we can choose this set at any moment anew and the additional information is no longer needed. As will become clear, this significantly reduces the SDP's number of states and, thus, its memory requirements.

For any state $P \in \mathcal{P}$, define the set of *eligible* tasks $E(P)$ as the set of tasks which, in principle, could be active. Formally, $E(P) = \{i \in N \setminus P \mid j \in P \text{ for all } (j, i) \in A\}$. A possible *action* then is to work on a set of tasks $\varphi \subseteq E(P)$. Choosing $\varphi = \emptyset$ when $P \neq N$ corresponds to project abandonment: we never execute the remaining tasks $N \setminus P$ and the project transitions to an additional terminal state that we simply call '*abandoned project*'. Section 4.3 discusses the changes if abandonment is not allowed.

It follows from [3] that there is no loss in optimality when choosing actions at state transitions only. Either it is desirable to work on task $i \in E(P)$ in state

$P \in \mathcal{P}$ and we do so during the entire visit of state P, or it is not desirable and we do not work on task i at all in state P. Intuitively speaking, the exponential distribution's memoryless property implies that as long as no transition occurs, one obtains no additional information that could alter the desirability of working on task i. This considerably simplifies the problem since we can limit decision moments to a discrete set of state transitions, while, in general, decisions could be made at any time, i.e. continuously, during the project's execution.

Suppose we choose action $\varphi \subseteq E(P)$ in state $P \in \mathcal{P}$, then the next state transition occurs randomly depending on which of the subsequently active tasks $i \in \varphi$ completes first. By the memoryless property of the exponential distribution, the remaining duration of each active task $i \in \varphi$ is exponentially distributed with parameter λ_i. The time until the next transition is the minimum of these durations and, thus, is exponentially distributed with parameter $\Lambda_\varphi := \sum_{i\in\varphi} \lambda_i$. Moreover, the probability that $i \in \varphi$ completes first equals $\lambda_i/\Lambda_\varphi$ and is independent from the time until the next transition. Finally, it is easily verified that the expected discount rate until the next state transition equals $\Lambda_\varphi/(r + \Lambda_\varphi)$ and that the expected cash flow incurred during the state's visit equals $\sum_{i\in\varphi} c_i/(r + \Lambda_\varphi)$.

Example. Figure 2 displays the possible sets of past activities $P \in \mathcal{P}$ for the example project of Figure 1. The node "1,3", for instance, represents state $P = \{1, 3\}$. In addition, there is a schematic representation of the possible transitions in the initial state. The square represents a decision node, indicating that the possible actions are the subsets of $E(\emptyset) = \{1, 2\}$. After choosing action $\varphi = \{1, 2\}$, the chance node (circle) indicates that the next transition occurs randomly depending on whether task 1 or 2 completes first.

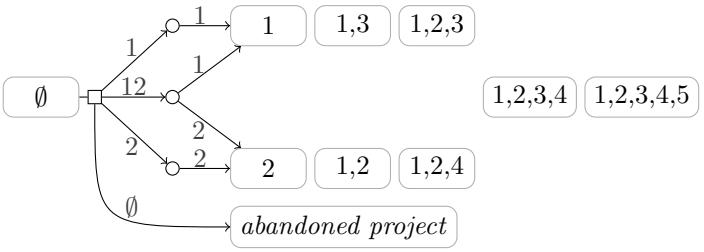

Figure 2: Illustration of states and transitions.

Define the *value function* $V(\cdot)$ as the mapping that assigns to each state $P \in \mathcal{P}$ the maximum eNPV that is attainable by the remaining activities when the tasks in P are past. It then follows from [7] that $V(\cdot)$ satisfies the *Optimality Equation*

$$V(P) = \max_{\varphi\subseteq E(P)} \left\{ \frac{\sum_{i\in\varphi} \lambda_i V(P_i) + c_i}{r + \sum_{i\in\varphi} \lambda_i} \right\} \tag{1}$$

and that choosing actions which attain the maximum constitute an optimal policy. Here, and in the sequel, we denote $P_i := P \cup \{i\}$, $P_{ij} := P \cup \{i, j\}$ and so forth. Since every task completion increases the cardinality of P with one unit, a backward recursion with boundary condition $V(N) = 0$ identifies $V(P)$. Our goal is to compute $V(\emptyset)$ and to identify a policy which attains this maximum.

Let $m := \max_{P \in \mathcal{P}} |E(P)|$ denote the maximum number of tasks that can be active in parallel, then it is not difficult to show that $|\mathcal{P}|$ is bounded above by $\sum_{i=0}^{m} \binom{n}{i}$; see [4]. Indeed, P is completely determined by $E(P)$ and $\binom{n}{i}$ bounds the number of states having $|E(P)| = i$. In fact, by the binomial theorem, $(1+n)^m = \sum_{i=0}^{m} \binom{m}{i} n^i \geq \sum_{i=0}^{m} n^i \geq \sum_{i=0}^{m} \binom{n}{i}$ and the number of states is polynomial in n for a fixed value of m. Since the completion of every subset $\varphi \subseteq E(P)$ leads to a feasible state $P \cup \varphi$, we also have $|\mathcal{P}| \geq 2^m$, implying that the number of states is exponential in m. If preemption is not allowed, we need to include the set of active tasks in each state [8]. Since all $Y \subseteq E(P)$, i.e. $2^{|E(P)|}$ subsets for each $P \in \mathcal{P}$, are possible sets of active tasks, the number of states for the non-preemptive case is bounded above by $\sum_{i=0}^{m} \binom{n}{i} 2^i$ and bounded below by 3^m. This suggests that the preemptive case is computationally more tractable than the non-preemptive one.

4 Structural properties

In the first part of this section, we characterize an optimal preemptive policy with a structure that facilitates the recursive computation of a state's value. Next, we use this structure to show how this optimal preemptive solution solves the non-preemptive case as well. Finally, we discuss the changes if project abandonment is not allowed. We will frequently use the following simple equivalence between (in)equalities:

Lemma 1. *For any $u, v \in \mathbb{R}$ and $w, \lambda \in \mathbb{R}_{>0}$ holds that*

$$v \geq \frac{u}{w} \iff \frac{u + \lambda v}{w + \lambda} \geq \frac{u}{w} \iff v \geq \frac{u + \lambda v}{w + \lambda},$$

and likewise for "$=$" or "$\leq$" instead of "$\geq$".

4.1 Structure of an optimal preemptive policy

Remember that c_i is the cash flow per time unit for working on task $i \in N$. $V(P_i)$, in turn, equals the value when i completes, while λ_i is the rate at which the task completes. Consequently, we can interpret $\lambda_i V(P_i)$ as the value per time unit generated by the possible completion of task i. Now define

$$\psi_i(P) := \frac{1}{\lambda_i}(\lambda_i V(P_i) + c_i) = V(P_i) + \frac{c_i}{\lambda_i},$$

then, since $1/\lambda_i$ equals task i's expected duration, $\psi_i(P)$ reflects the total expected value of working on task i. Finally, denote

$$\varphi^\star(P) := \{i \in E(P) \mid \psi_i(P) \geq V(P)\},$$

then the next result states that we should work on task i whenever the total expected value of working on i is not less than the current value $V(P)$.

Proposition 1. *Let $P \in \mathcal{P}$, then $\varphi^\star(P)$ is an optimal action in state P.*

Proof. Given a state $P \in \mathcal{P}$, consider an eligible task $i \in E(P)$ and action $\varphi \subseteq E(P)$ such that $i \notin \varphi$. From the first equivalence in Lemma 1,

$$\psi_i(P) \geq \frac{\sum_{j \in \varphi} \lambda_j \psi_j(P)}{r + \sum_{j \in \varphi} \lambda_j} \tag{2}$$

if and only if

$$\frac{\sum_{j\in\varphi}\lambda_j\psi_j(P)+\lambda_i\psi_i(P)}{r+\sum_{j\in\varphi}\lambda_j+\lambda_i}\geq\frac{\sum_{j\in\varphi}\lambda_j\psi_j(P)}{r+\sum_{j\in\varphi}\lambda_j}.$$

By Optimality Equation (1), $V(P)$ equals the maximum of Inequality (2)'s right-hand side over all actions $\varphi\subseteq E(P)$ and, thus, working on task i never decreases value if $\psi_i(P)\geq V(P)$. Similarly, from Lemma 1's second equivalence, an optimal action never includes task i if $\psi_i(P)<V(P)$. □

While the result of Proposition 1 is intuitively appealing, it does not help from a computational point of view. Indeed, when searching for the optimal action in state P we obviously do not know the value $V(P)$ yet. The next result expresses $\varphi^\star(P)$ in terms of the value function evaluated at sets of past activities with cardinality $|P|+1$. As such, it is useful for the backward recursion. For any $P\in\mathcal{P}$ and $i\in E(P)$, define

$$\tau_i(P):=\frac{\sum_{j\in E(P):\psi_j(P)>\psi_i(P)}\lambda_jV(P_j)+c_j}{r+\sum_{j\in E(P):\psi_j(P)>\psi_i(P)}\lambda_j},$$

then $\tau_i(P)$ can be interpreted as the value of choosing action $\varphi=\{j\in E(P)\mid\psi_j(P)>\psi_i(P)\}$ in Optimality Equation (1). The intuition behind Theorem 1 is as follows. Since $\psi_i(P)$ reflects the value of working on i, we prefer to work on tasks having higher $\psi_i(P)$. The result then states that we want to 'add' task i to the set of active tasks if working on i does not decrease value compared to the situation where we only work on tasks j with $\psi_j(P)>\psi_i(P)$.

Theorem 1. *Let $P\in\mathcal{P}$, then $\varphi^\star(P)=\{i\in E(P)\mid\psi_i(P)\geq\tau_i(P)\}$.*

Proof. It suffices to show that, for each $P\in\mathcal{P}$ and $i\in E(P)$, it holds that $\psi_i(P)\geq V(P)$ if and only if $\psi_i(P)\geq\tau_i(P)$. The result that $\psi_i(P)\geq V(P)$ implies $\psi_i(P)\geq\tau_i(P)$ immediately follows from the fact that $V(P)\geq\tau_i(P)$. Indeed, $\{j\in E(P)\mid\psi_j(P)>\psi_i(P)\}$ is feasible for Optimality Equation (1) and leads to value $\tau_i(P)$.

For the other direction, i.e. that $\psi_i(P)\geq\tau_i(P)$ implies $\psi_i(P)\geq V(P)$, define the set $L:=\{j\in E(P)\mid\psi_j(P)\geq\tau_j(P)\}$ and a permutation ρ on L such that $\psi_{\rho(l)}\geq\psi_{\rho(l+1)}$ for all $l=1,\ldots,|L|-1$. Induction on l will show that $\psi_j(P)\geq V(P)$ for all $j\in L$.

Suppose that $\psi_{\rho(1)}(P)<V(P)$, then, by definition of ρ, $\psi_j(P)<V(P)$ for all $j\in L$. Moreover, by definition of L and the first part, $\psi_j(P)<V(P)$ for all $j\in E(P)\setminus L$ as well. Proposition 1 then implies that $\varphi^\star(P)=\emptyset$ and $V(P)=0$. Together with $0=\tau_{\rho(1)}\leq\psi_{\rho(1)}(P)$ this gives the contradiction: $0\leq\psi_{\rho(1)}<V(P)=0$. Consequently, $\psi_{\rho(1)}(P)\geq V(P)$.

Now take arbitrary $k\in\{2,\ldots,|L|\}$ and assume $\psi_{\rho(l)}(P)\geq V(P)$ for $l<k$. The induction step consists of showing that $\psi_{\rho(k)}(P)\geq V(P)$ as well. If not, then similar reasoning as above yields $\psi_{\rho(l)}<V(P)$ for all $l=k+1,\ldots,|L|$ and $\psi_j<V(P)$ for $j\in E(P)\setminus L$. Since $\psi_{\rho(k)}<\psi_{\rho(k-1)}$, the induction hypothesis and Proposition 1 imply $V(P)=\tau_{\rho(k)}(P)$ and, together with $\tau_{\rho(k)}\leq\psi_{\rho(k)}(P)$, this gives a contradiction. Consequently, $\psi_{\rho(k)}(P)\geq V(P)$. □

4.2 Optimality of a non-preemptive policy

Not exercising preemption can be seen as a monotonicity in the actions: if we decide to work on task $i \in E(P)$ in state $P \in \mathcal{P}$, we want to work on it in any successor state P_j with $j \in E(P) \setminus \{i\}$ as well. Such monotonicity would imply that an optimal preemptive solution solves the non-preemptive case as well. This is useful for at least two reasons. First, from a computational point of view, the preemptive case is easier to deal with (see Section 3). In addition, from a managerial point of view, it is interesting to know under which circumstances it is not useful to consider preemption.

The remainder of this section shows that a monotone preemptive policy is indeed optimal. First, Lemma 2 gives the intuitively obvious result that, since $c_i \leq 0$ for $i \in N \setminus \{n\}$, the completion of an intermediary task does not decrease the value function. Lemma 3, in turn, delivers the key step by showing that the change in value thanks to a task's completion does not decrease if another task finishes first. It proves that $V(\cdot)$ is supermodular, a property underlying numerous monotonicity proofs [6, 9]. Since the proof for Lemma 2 is relatively straightforward and the one for Lemma 3 rather lengthy, they are both included in appendix.

Lemma 2. *Let $P \in \mathcal{P}$ with $n \notin E(P)$, then for any $i \in E(P)$: $V(P_i) \geq V(P)$.*

Lemma 3. *Let $P \in \mathcal{P}$ with $|E(P)| \geq 2$, then for any $i, j \in E(P)$ with $i \neq j$: $V(P_{ij}) - V(P_j) \geq V(P_i) - V(P)$.*

Proposition 2. *Let $P \in \mathcal{P}$ with $|E(P)| \geq 2$, then for any $i, j \in E(P)$ with $i \neq j$: $i \in \varphi^\star(P)$ implies $i \in \varphi^\star(P_j)$.*

Proof. By definition of $\varphi^\star(P)$ and $\psi_i(P)$, it holds that $i \in \varphi^\star(P)$ if and only if $V(P_i) - V(P) \geq -c_i/\lambda_i$. This implies $i \in \varphi^\star(P_j)$ as well since, by Lemma 3, $V(P_{ij}) - V(P_j) \geq V(P_i) - V(P) \geq -c_i/\lambda_i$. □

Theorem 2. *$\varphi^\star(\cdot)$ fully determines an optimal non-preemptive policy.*

Proof. From Proposition 2, $\varphi^\star(\cdot)$ never interrupts a task which was active in a previous state. Consequently, the resulting policy is non-preemptive and, thus, is optimal for the non-preemptive case as well. □

4.3 Project abandonment

From Lemma 2, $V(P) > 0$ for all $P \in \mathcal{P}$ if $V(\emptyset) > 0$. Thus, either we never abandon the project, or we do not initiate it. This result was also proposed by Sobel et al. [8, Proposition 2]; Lemma 2 gives an alternative proof.

One could think of situations, however, where we want to know the maximum eNPV, even if it is negative. This means that choosing $\varphi = \emptyset$ in Optimality Equation (1) is forbidden. In order to deal with this situation, let $P \in \mathcal{P}$ and take arbitrary

$$i^\star(P) \in \arg\max_{i \in E(P)} \left\{ \frac{\lambda_i V(P_i) + c_i}{r + \lambda_i} \right\}.$$

Next, consider the action

$$\bar{\varphi}^\star(P) := \begin{cases} i^\star(P) & \text{if } \psi_i(P) \leq 0 \text{ for all } i \in E(P); \\ \varphi^\star(P) & \text{otherwise.} \end{cases}$$

It then follows that $\bar{\varphi}^\star(\cdot)$ describes a policy where we perform the tasks sequentially as long as the value function is negative and switch to $\varphi^\star(\cdot)$ as soon as the value function becomes positive. The next proposition shows that $\bar{\varphi}^\star(\cdot)$ is optimal if it is not allowed to abandon the project.

Proposition 3. *Let $P \in \mathcal{P}$, then $\bar{\varphi}^\star(P)$ is an optimal action in state P if choosing $\varphi = \emptyset$ is forbidden. Moreover, $\bar{\varphi}^\star(\cdot)$ is non-preemptive.*

Proof. If $\psi_i(P) \leq 0$ for all $i \in E(P)$, then

$$\psi_i(P) = \frac{\lambda_i V(P_i) + c_i}{\lambda_i} \leq \frac{\lambda_i V(P_i) + c_i}{r + \lambda_i}$$

and the optimality of initiating task $i^\star(P)$ follows from the definition of $i^\star(P)$ in combination with the first equivalence of Lemma 1.

On the other hand, if $\psi_i(P) > 0$ for some $i \in E(P)$, then $V(P) > 0$ and applying Lemma 2 yields $V(P') > 0$ for all successor states P' of P as well. Consequently, no abandonment will occur in the remainder of the project and Proposition 1 proves the optimality of $\varphi^\star(P)$.

Finally, to see that $\bar{\varphi}^\star(\cdot)$ is non-preemptive, first note that $\bar{\varphi}^\star(\cdot)$ only initiates a single activity if $\psi_i(P) \leq 0$ for all $i \in E(P)$; this task will be finished in the next state and no preemption occurs. Next, if $\psi_i(P) > 0$ for some $i \in E(P)$, non-preemption is guaranteed by Proposition 2. □

5 Algorithm and performance

Algorithm 1 presents our new procedure to identify a project's maximum eNPV for both the non-preemptive and preemptive case. The set $\mathcal{P}_k$ collects all states $P \in \mathcal{P}$ having exactly $k \in \{0, 1, \ldots, n\}$ completed tasks and, thus, $(\mathcal{P}_0, \mathcal{P}_1, \cdots, \mathcal{P}_n)$ forms a partition of $\mathcal{P}$. As suggested by [5], this partitioning of $\mathcal{P}$ into different *stages* significantly reduces the memory requirements since, after having evaluated the value function for all states in $\mathcal{P}_{k-1}$, we no longer need the information in $\mathcal{P}_k$.

Algorithm 1 Determine the maximum eNPV.

input: $N, A, (c_i, \lambda_i)_{i \in N}, r$
$V(N) \leftarrow 0$; $\varphi^\star(N) \leftarrow \emptyset$; $\mathcal{P}_n \leftarrow \{N\}$
for stage $k \leftarrow n$ **downto** 1 **do**
    based on $\mathcal{P}_k$, determine states in $\mathcal{P}_{k-1}$
    for states $P \in \mathcal{P}_{k-1}$ **do**
        determine $\varphi^\star(P)$ and $V(P)$
    end for
    free memory occupied by $\mathcal{P}_k$
end for
return $V(\emptyset)$

In the remainder, we compare the performance of Algorithm 1 with the procedure of Creemers et al. [2] based on a number of test instances grouped by their value of n and *order strength* (OS). The latter measures the project network's density and equals the ratio of the number of pairs in A, i.e. $|A|$, and

the maximum number of such pairs, i.e. $n(n-1)/2$. We reuse the dataset of [2], who generated 30 instances for each combination of $n \in \{10, 20, \ldots, 120\}$ and $OS \in \{0.8, 0.6, 0.4\}$. We follow [2] by not allowing for project abandonment.

All experiments were performed with an Intel Core i7-4790 processor with 3.60 GHz CPU speed and we used an upper bound of 1 GB of RAM. Table 1 shows the average CPU time in seconds and the number of instances solved without violating the memory constraint. Tables 2-3, in turn, display the average total number of states as well as the average maximal fraction of these states kept in memory at the same time. This fraction measures how effectively the state space's partitioning mitigates memory requirements. All averages are taken over solved instances only.

Given the discussion in Section 3, it is not surprising that OS is a major determinant of performance for both methods since a sparser network allows for more tasks executable in parallel. Our method clearly improves on the procedure of [2] with several orders of magnitude, both in memory usage and CPU time. To explain the reduced memory requirements, remember from Section 3 that the bounds on the number of states for our SDP are considerably lower than the ones for the method of [2]. It is interesting to note that the fraction of states in memory for our method is not only lower, but is also less sensitive to the order strength. This results from the fact that, contrary to the method of [2], Algorithm 1's partitioning of the state space does not depend on the number of tasks executable in parallel. Finally, to explain the reduction in CPU time, note that, in addition to the decrease in computational effort thanks to the reduced number of states, Theorem 1 enables to determine $\varphi^\star(P)$ without enumerating all subsets of $E(P)$. Since memory usage rather than CPU time still constitutes the algorithm's bottleneck, further research should focus on identifying additional properties which could mitigate the algorithm's memory usage.

Table 1: Average CPU time in seconds and number (#) of solved instances out of 30.

	Creemers et al. [2]						Algorithm 1					
	OS=0.80		OS=0.60		OS=0.40		OS=0.80		OS=0.60		OS=0.40	
n	CPU	#	CPU	#	CPU	#	CPU	#	CPU	#	CPU	#
10	0.00	30	0.00	30	0.00	30	0.00	30	0.00	30	0.00	30
20	0.00	30	0.01	30	0.27	30	0.00	30	0.00	30	0.00	30
30	0.00	30	0.19	30	15.71	30	0.00	30	0.00	30	0.02	30
40	0.02	30	3.84	30	1,213.43	29	0.00	30	0.01	30	0.31	30
50	0.08	30	50.70	30	11,287.77	2	0.00	30	0.06	30	3.95	30
60	0.42	30	1,288.93	30		0	0.01	30	0.29	30	58.96	30
70	1.63	30	5,230.08	17		0	0.01	30	1.52	30	420.62	26
80	6.06	30	25,628.35	2		0	0.03	30	5.72	30		0
90	27.60	30		0		0	0.06	30	38.58	30		0
100	98.09	30		0		0	0.12	30	151.44	30		0
110	590.56	30		0		0	0.30	30	501.76	27		0
120	2,495.16	26		0		0	1.18	30	1,473.39	10		0

Table 2: Average total number (#) of states and average maximal fraction (%) of states in memory for the method of Creemers et al. [2].

	Creemers et al. [2]					
	OS=0.80		OS=0.60		OS=0.40	
n	# states	%	# states	%	# states	%
10	71	40	206	52	695	57
20	484	35	4,006	46	55,016	58
30	1,995	29	49,388	42	1,560,364	50
40	7,860	29	534,014	41	47,072,515	48
50	26,667	31	4,346,215	41	348,684,057	23
60	92,003	32	42,278,506	41		
70	286,831	32	165,870,016	36		
80	829,741	28	603,402,153	20		
90	2,596,419	31				
100	6,868,100	34				
110	24,235,588	34				
120	112,181,874	25				

Table 3: Average total number (#) of states and average maximal fraction (%) of states in memory for Algorithm 1.

	Algorithm 1					
	OS=0.80		OS=0.60		OS=0.40	
n	# states	%	# states	%	# states	%
10	22	29	41	32	84	33
20	88	21	330	21	1,620	24
30	254	14	1,898	16	17,096	17
40	662	12	9,480	13	193,848	14
50	1,544	11	40,379	12	1,659,705	12
60	3,564	10	175,288	11	13,790,968	11
70	7,754	10	727,948	10	77,917,666	9
80	16,364	8	2,259,476	8		
90	34,057	8	9,845,427	9		
100	66,670	8	31,268,023	8		
110	146,910	8	90,918,894	7		
120	515,785	7	239,953,385	5		

References

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

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

[3] E. A. Feinberg. Continuous time discounted jump Markov decision pro-

cesses: a discrete-event approach. *Mathematics of Operations Research*, 29 (3):492–524, 2004.

[4] E. Gutin, D. Kuhn, and W. Wiesemann. Interdiction games on Markovian PERT networks. *Management Science*, 61(5):999–1017, 2014.

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

[6] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[7] R. F. Serfozo. Technical note – an equivalence between continuous and discrete time Markov decision processes. *Operations Research*, 27(3):616–620, 1979.

[8] M. J. Sobel, J. G. Szmerekovsky, and V. Tilson. Scheduling projects with stochastic activity duration to maximize expected net present value. *European Journal of Operational Research*, 198(3):697 – 705, 2009.

[9] D. M. Topkis. Minimizing a submodular function on a lattice. *Operations Research*, 26(2):305–321, 1978.

[10] W. Wiesemann and D. Kuhn. The stochastic time-constrained net present value problem. In *Handbook on Project Management and Scheduling*, volume 2, pages 753–780. Springer, 2015.

Appendix – Proof of Lemma 2 and Lemma 3

Proof of Lemma 2. The proof uses induction on the cardinality of P. First, consider the situation where $|P| = n-2$ such that $P = N \setminus \{i, n\}$, $E(P) = \{i\}$, $P_i = N \setminus \{n\}$ and $E(P_i) = \{n\}$. It follows that

$$V(P) = \max\left\{0; \frac{\lambda_i V(P_i) + c_i}{r + \lambda_i}\right\} \text{ and } V(P_i) = \max\left\{0; \frac{c_n}{r + \lambda_n}\right\}.$$

From $c_i \leq 0 < c_n$ then follows that $V(P) \leq V(P_i)$.

Next, take arbitrary $P \in \mathcal{P}$ with $|P| < n-2$. The induction hypothesis states that $V(P'_k) \geq V(P')$ for any $P' \in \mathcal{P}$ with $|P'| = |P| + 1$ and any $k \in E(P')$. This implies $V(P) \leq V(P_i)$ since

$$\begin{aligned} V(P) &\leq \max_{\varphi \subseteq E(P)} \left\{ \frac{\lambda_i V(P_i)\mathbb{I}(i \in \varphi) + \sum_{k \in \varphi \setminus \{i\}} \lambda_k V(P_k) + c_k}{\lambda_i \mathbb{I}(i \in \varphi) + r + \sum_{k \in \varphi \setminus \{i\}} \lambda_k} \right\} \\ &\leq \max_{\varphi \subseteq E(P_i) \cup \{i\}} \left\{ \frac{\lambda_i V(P_i)\mathbb{I}(i \in \varphi) + \sum_{k \in \varphi \setminus \{i\}} \lambda_k V(P_{ik}) + c_k}{\lambda_i \mathbb{I}(i \in \varphi) + r + \sum_{k \in \varphi \setminus \{i\}} \lambda_k} \right\} = V(P_i). \end{aligned}$$

Here, the indicator function $\mathbb{I}(\cdot)$ yields 1 when the condition within parentheses is satisfied and 0 otherwise. The first inequality follows from $c_i \leq 0$, while the second results from the induction hypothesis and $E(P) \subseteq E(P_i) \cup \{i\}$. Lemma 1 and Optimality Equation (1), finally, imply the equality. □

Proof of Lemma 3. The proof is by induction on the cardinality of P. First, assume P has maximal cardinality in the sense that no other $P' \in \mathcal{P}$ exists for which $|E(P')| \geq 2$ and $|P'| > |P|$. Denote this maximal cardinality $|P|$ by q. Call i and j the two[1] activities in $E(P)$ and note[2] that $E(P_j) = \{i\}$.

Suppose $V(P_j) = 0$, then Lemma 2 implies $V(P) = 0$ and $V(P_{ij}) - V(P_j) \geq V(P_i) - V(P)$ follows from $V(P_{ij}) \geq V(P_i)$. For $V(P_j) > 0$, in turn, $E(P_j) = \{i\}$ implies

$$V(P_j) = \frac{\lambda_i V(P_{ij}) + c_i}{r + \lambda_i} \iff V(P_{ij}) - V(P_j) = \frac{rV(P_j) - c_i}{\lambda_i}.$$

On the other hand, $i \in E(P)$ implies

$$V(P) \geq \frac{\lambda_i V(P_i) + c_i}{r + \lambda_i} \iff V(P_i) - V(P) \leq \frac{rV(P) - c_i}{\lambda_i}.$$

$V(P_j) \geq V(P)$ then yields $V(P_{ij}) - V(P_j) \geq V(P_i) - V(P)$.

Next, take an arbitrary $P \in \mathcal{P}$ with $|E(P)| \geq 2$ and $|P| < q$. The induction hypothesis is that, for all $P' \in \mathcal{P}$ having $|E(P')| \geq 2$ and $|P'| > |P|$, it holds for any $k, l \in E(P')$, $k \neq l$, that $V(P'_{kl}) - V(P'_l) \geq V(P'_k) - V(P')$. The inductive step, in turn, consists of showing that, for any $i, j \in E(P)$, $i \neq j$, this implies $V(P_{ij}) - V(P_j) \geq V(P_i) - V(P)$. Define $\alpha := V(P_i) + V(P_j) - V(P_{ij}) - V(P)$, then it suffices to show that $\alpha \leq 0$.

For notational convenience, denote

$$\begin{array}{llll} \varphi_i := \varphi^\star(P_i) & \varphi_j := \varphi^\star(P_j) & \varphi_\cup := \varphi_i \cup \varphi_j & \varphi_{i\setminus j} := (\varphi_i \setminus \varphi_j) \setminus \{j\} \\ \delta_i := \mathbb{I}(i \in \varphi_j) & \delta_j := \mathbb{I}(j \in \varphi_i) & \varphi_\cap := \varphi_i \cap \varphi_j & \varphi_{j\setminus i} := (\varphi_j \setminus \varphi_i) \setminus \{i\} \end{array}$$

It then follows that

$$\begin{aligned} V(P_i) &= \frac{\sum_{k\in\varphi_i} \lambda_k V(P_{ik}) + c_k}{r + \sum_{k\in\varphi_i} \lambda_k} \\ &= \frac{\sum_{k\in\varphi_i} (\lambda_k V(P_{ik}) + c_k) + \sum_{k\in\varphi_{j\setminus i}} \lambda_k V(P_i) + \delta_i \lambda_i V(P_i)}{r + \sum_{k\in\varphi_\cup} \lambda_k}. \end{aligned}$$

Here, the first equation follows from the optimality of $\varphi^\star(P_i)$, while the second equation results from Lemma 1. Similarly, by interchanging the role of i and j,

$$V(P_j) = \frac{\sum_{k\in\varphi_j} (\lambda_k V(P_{jk}) + c_k) + \sum_{k\in\varphi_{i\setminus j}} \lambda_k V(P_j) + \delta_j \lambda_j V(P_j)}{r + \sum_{k\in\varphi_\cup} \lambda_k}.$$

Note that $(\varphi_\cup) \setminus \{i, j\}$ is a feasible action in state P_{ij} since $(E(P_i) \cup E(P_j)) \setminus \{i, j\} \subseteq E(P_{ij})$. Consequently,

$$V(P_{ij}) \geq \frac{\sum_{k\in\varphi_\cup\setminus\{i,j\}} \lambda_k V(P_{ijk}) + c_k}{r + \sum_{k\in\varphi_\cup\setminus\{i,j\}} \lambda_k}$$

and thus, by Lemma 1,

$$V(P_{ij}) \geq \frac{\sum_{k\in(\varphi_i\cup\varphi_j)\setminus\{i,j\}} (\lambda_k V(P_{ijk}) + c_k) + \delta_i \lambda_i V(P_{ij}) + \delta_j \lambda_j V(P_{ij})}{r + \sum_{k\in\varphi_\cup} \lambda_k}.$$

[1] $|E(P)| = 2$, if not: $|E(P_i)| \geq 2$ and $|P_i| > |P|$ for any $i \in E(P)$.

[2] Maximal cardinality of P implies $|E(P_j)| < 2$, while $i \in E(P)$ implies $i \in E(P_j)$.

Finally, $\varphi_\cap \cup \{i,j\}$ is a feasible action in state P since $E(P) = (E(P_i) \cap E(P_j)) \cup \{i,j\}$. Consequently,

$$V(P) \geq \frac{\sum_{k\in\varphi_\cap}(\lambda_k V(P_k) + c_k) + \delta_i(\lambda_i V(P_i) + c_i) + \delta_j(\lambda_j V(P_i) + c_j)}{r + \sum_{k\in\varphi_\cap}\lambda_k + \delta_i\lambda_i + \delta_j\lambda_j}$$

and thus, by Lemma 1,

$$\begin{aligned} V(P) \geq \frac{1}{r + \sum_{k\in\varphi_\cup}\lambda_k}\Big(& \sum_{k\in\varphi_\cap}(\lambda_k V(P_k) + c_k) + \sum_{k\in\varphi_{i\setminus j}}\lambda_k V(P) \\ & + \sum_{k\in\varphi_{j\setminus i}}\lambda_k V(P) + \delta_i(\lambda_i V(P_i) + c_i) + \delta_j(\lambda_j V(P_i) + c_j)\Big). \end{aligned}$$

Substituting the (in)equalities for $V(P_i)$, $V(P_j)$, $V(P_{ij})$ and $V(P)$ into α, we obtain

$$\begin{aligned} \alpha \leq \frac{1}{r + \sum_{k\in\varphi_\cup}\lambda_k}\Big(& \sum_{k\in\varphi_\cap}\lambda_k\left(V(P_{ik}) + V(P_{jk}) - V(P_{ijk}) - V(P_k)\right) \\ & + \sum_{k\in\varphi_{i\setminus j}}\lambda_k\left(V(P_{ik}) + V(P_j) - V(P_{ijk}) - V(P)\right) \\ & + \sum_{k\in\varphi_{j\setminus i}}\lambda_k\left(V(P_i) + V(P_{jk}) - V(P_{ijk}) - V(P)\right) \\ & + \delta_i\lambda_i\left(V(P_i) + V(P_{ij}) - V(P_{ij}) - V(P_i)\right) \\ & + \delta_j\lambda_j\left(V(P_{ij}) + V(P_j) - V(P_{ij}) - V(P_j)\right)\Big). \end{aligned}$$

From the induction hypothesis, we respectively have for each $k \in \varphi_\cap$, $k \in \varphi_{i\setminus j}$ and $k \in \varphi_{j\setminus i}$ that

$$\begin{aligned} & V(P_{ik}) + V(P_{jk}) - V(P_{ijk}) - V(P_k) \leq 0; \\ & V(P_{ik}) - V(P_{ijk}) \leq V(P_i) - V(P_{ij}); \\ & V(P_{jk}) - V(P_{ijk}) \leq V(P_j) - V(P_{ij}). \end{aligned}$$

Substituting these inequalities into the inequality for α yields

$$\begin{aligned} \alpha &\leq \frac{1}{r + \sum_{k\in\varphi_\cup}\lambda_k}\Big(\sum_{k\in\varphi_{i\setminus j}}\lambda_k\left(V(P_i) + V(P_j) - V(P_{ij}) - V(P)\right) \\ &\qquad + \sum_{k\in\varphi_{j\setminus i}}\lambda_k\left(V(P_i) + V(P_j) - V(P_{ij}) - V(P)\right)\Big) \\ &= \alpha\frac{\sum_{k\in\varphi_{i\setminus j}}\lambda_k + \sum_{k\in\varphi_{j\setminus i}}\lambda_k}{r + \sum_{k\in\varphi_\cup}\lambda_k}. \end{aligned}$$

The fact that $0 \leq \frac{\sum_{k\in\varphi_{i\setminus j}}\lambda_k + \sum_{k\in\varphi_{j\setminus i}}\lambda_k}{r + \sum_{k\in\varphi_\cup}\lambda_k} < 1$ then proves that $\alpha \leq 0$. □

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