Setting gates for activities in the stochastic project scheduling problem through the cross entropy methodology

Illana Bendavid · Boaz Golany

Abstract This paper addresses the problem of scheduling activities in projects with stochastic activity durations. The aim is to determine for each activity a gate—a time before it the activity cannot begin. Setting these gates is analogous to setting inventory levels in the news vendor problem. The resources required for each activity are scheduled to arrive according to its gate. Since activities' durations are stochastic, the start and finish time of each activity is uncertain. This fact may lead to one of two outcomes: (1) an activity is ready to start its processing as all its predecessors have finished, but it cannot start because the resources required for it were scheduled to arrive at a later time. (2) The resources required for the activity have arrived and are ready to be used but the activity is not ready to start because of precedence constraints. In the first case we will incur a "holding" cost while in the second case, we will incur a "shortage" cost. Our objective is to set gates so as to minimize the sum of the expected holding and shortage costs. We employ the Cross-Entropy method to solve the problem. The paper describes the implementation of the method, compares its results to various heuristic methods and provides some insights towards actual applications.

1 Introduction

The project scheduling problem has given rise to an extensive literature that addresses numerous variations of the basic problem. The various formulations of the problem differ from each other by the characterization of the activities durations, the existence of resource constraints and, most importantly, by their objective functions.

Basic techniques for project scheduling have evolved since the 1950s. A detailed description of these methodologies can be found in textbooks such as Elmaghraby (1977), Demeulemeester and Herroelen (2002), and Shtub et al. (2005). Two of the most common techniques are the Critical Path Method (CPM) which assumes deterministic activity times and

I. Bendavid (✉) · B. Golany
Faculty of Industrial Engineering and Management, Technion–Israel Institute of Technology,
Haifa 32000, Israel
e-mail: illana@tx.technion.ac.il

Program Evaluation and Review Technique (PERT) which assumes random activity durations (see Krishnamoorthy 1968). These methods share the same objective—makespan minimization, which has been the most popular objective in the project scheduling domain. This objective was first used for projects subject to precedence constraints only (see Fazar 1959; Kolish and Padman 2001; Krishnamoorthy 1968; Malcom et al. 1959), then for projects subject to precedence and resource constraints (see reviews in Kolish and Padman (2001) and Herroelen et al. (1998) on deterministic project scheduling and the review in Herroelen and Leus (2005) on the stochastic problem).

Critical Chain/Buffer Management (CC/BM) is a more recent method (see Goldratt 1997 and Rand 2000). CC/BM identifies the critical chain as the longest sequence of activities (determined simultaneously by the precedence relations and resources availability) that determines the overall completion time of the project. Project control in CC/BM is performed via time buffers which are added in and around the critical chain. These buffers are analogous to inventory buffers in production line. In the latter we wish to prevent "starvation" of succeeding machine or "blockage" of preceding machine while in the project scheduling environment we wish to avoid changes in the plan when some activities are delayed or are completed earlier than planned.

In recent years, new methodologies have evolved that maximize the net present value (NPV) of projects with deterministic durations—see the reviews in Kimms (2001) and Herroelen et al. (1997). Maximization of NPV in stochastic settings was considered, to the best of our knowledge, only by Buss and Rosenblatt (1997) and by Sobel et al. (2009). Analogous to the CC/BM methodology, this methodology also attempts to facilitate the control over the project schedule. In particular, when maximizing NPV, one finds that starting activities as soon as possible (according to precedence constraints) is not always optimal.

Lately, a new approach to project scheduling under uncertainty has emerged. This approach consists of determining for each activity a *gate*—a time before which the activity cannot begin. This approach stems from earlier works on scheduling of manufacturing jobs on a single machine by determining for each job a start time that maximizes profit (see Elmaghraby et al. 2000; Elmaghraby 2001). These works include the deterministic case, where the release time of each activity (i.e., the time in which the activity can start) is known in advance, as well as the stochastic case where the release times are independent random variables, approximated by a Normal distribution. Both cases are solved by formulating the problem as a dynamic programming model. Elmaghraby (2001) also considers the specific case where all activities must be released at the same time, *en masse*.

Trietsch (2006a, 2006b) motivates the need for gates in an outsourcing environment where resources are booked well in advance and are required to be ready for an activity at a predetermined time. If the resource is ready but the activity is not, then a cost per time unit is incurred. Trietsch develops a "criticality index" for each activity and then suggests that the gates should be driven from these indexes. However, he doesn't offer a specific method to determine these gates.

The goal of this paper is to develop a methodology that will determine a gate for each activity so as to minimize the expected penalty costs. The idea behind this approach is that in an environment with stochastic durations we cannot know for sure the start time and the finish time of each activity; this fact may lead to one of two outcomes: (1) an activity is potentially ready to start its processing since all its predecessors have finished, but it cannot actually start because the resources required for it were planned to arrive at a subsequent time. (2) The resources required for an activity have arrived and are ready to be used but the activity is not ready to start its processing because of precedence constraints. In both cases penalty costs are incurred. In the first case, the penalty is similar to an inventory holding

cost incurred by the fact that the activity is ready sooner than planned. The holding cost is incurred since we have invested money to execute the preceding activities sooner than needed. It can also result from an indirect cost in cases where the products of preceding activities deteriorate if not used immediately or within a specific time interval. In the second case, the penalty is similar to an inventory shortage cost incurred by the fact that the activity is ready later than planned. Again, this shortage cost can result from an alternative cost due to the fact that we have invested money to order the resources sooner than needed. It can also result from an indirect cost in cases where the resources deteriorate if not used immediately or within a specific time interval. Also, the entire project has a due date (generally imposed exogenously) that is also the due date of last activities of the project. Failure to meet this due-date results in a "shortage" penalty which is typically larger than the shortage penalties which are associated with the individual activities of the project. The method will determine the vector of gates that minimize the expected sum of holding and shortage costs. Similarly to the CC/BM and the NPV maximization methodologies, our goal is to control the schedule of the project by determining start times for each activity with the idea that starting activities as soon as possible is not always optimal.

The rest of this paper is organized as follows. In the next section we formulate the problem and present a dynamic programming formulation of the problem. In Sect. 3, we present the Cross-Entropy (CE) approach and its implementation to our particular problem. In Sect. 4, we present the comparison of CE results to four heuristic methods: an Early Start approach, a Late Start approach, a random search heuristic and a Simulated Annealing approach. In Sect. 5, we provide concluding remarks and some insights towards actual applications.

2 Problem description

2.1 Problem environment

First, we define the following notation:

n	the number of activities in the project
P_i	the set of predecessors of activity i
S_i	the set of successors of activity i
Y_i	the random variable that describes the duration of activity i
y_i	the realization of the random variable Y_i
F_i	the cumulative distribution function of the random variable Y_i
a_i	the minimal value of the random variable Y_i
b_i	the maximal value of the random variable Y_i
g_i	the gate of activity i (a decision variable)
t_i	the realization of the start time of activity i
h_i	the holding cost per unit time for activity i
p_i	the shortage cost per unit time for activity i
d	the due date for the entire project
s_i	the delay that has been realized for activity i ($s_i \geq 0$)
k_i	the time between the gate of activity i and the gate of activity $i+1$ (a decision variable)
$\ell[\sigma, \alpha, V(\tau)]$	the local income function: the expected cost associated with observing state σ, taking action α and gaining terminal reward $V(\tau)$ if the transition occurs to state τ

$V_i(g_i, s_i)$ the optimal expected cost from activity i to the end of the project,
given that we are in state (g_i, s_i)

We consider a project composed of n activities. These activities are linked by precedence relationships that reflect organizational and technological constraints. The links between activities produce a directed graph composed of paths that can be parallel to each other or connected by some activities. We define P_i and S_i to be the sets of predecessors and successors of activity i, respectively. Specific problem instances can be found, for example, in Herroelen et al. (1998).

We consider the case where the duration of each activity is a random variable. Each activity i has a duration Y_i, which is a discrete random variable with a realization y_i, a probability function and a cumulative distribution function F_i. It is assumed that the distribution of Y_i is bounded in the interval $\{a_i, a_i + 1, \ldots, b_i\}$ $(0 \leq a_i \leq b_i)$. The gates of activities are the decision variables. Once they are determined, the processing of an activity cannot start before its gate but it may start later (if its predecessors are delayed). The recursion that determines the starting time t_i of activity i whose gate was set at g_i and whose predecessors $j \in P_i$ have started at time t_j is given by:

$$t_i = \max_{j \in P_i}\{g_i, t_j + y_j\}.$$

As explained before, in such a stochastic environment, two outcomes are possible: (1) a specific activity is potentially ready to start (as its predecessors are done) but cannot actually start because the resources required for it were planned to arrive at a later time; (2) The resources required for a specific activity have arrived but the activity is not ready to start because one or more of its predecessors has not yet finished. In both cases penalty costs are incurred. In the first case, the penalty is similar to a holding cost since we "carry" the predecessors in the system as "inventory" during the waiting time for the resources. In the second case, the penalty is similar to a shortage cost since we cannot "supply" the resources which have already arrived with the "inventory" they need (that is, finished predecessors).

Consequently, we define h_i and p_i as the holding and shortage costs per unit time, respectively, for activity i. An ideal situation would be to finish processing all predecessors of activity i exactly at time g_i. Hence, if the processing of one of the predecessors of activity i ends before the time g_i, holding costs are incurred and if it ends after the time g_i, shortage costs are incurred. The entire project has a due date d (generally imposed by external agents) that is also the due date of the last activities of the project.

2.2 Dynamic programming formulation of the problem

We next consider the problem of determining gates of activities in a serial project, that is, where each activity has at most one predecessor and one successor. Without loss of generality, denote the activities $1, \ldots, n$ with $P_i = \{i - 1\}$ for $i = 2, \ldots, n$ and $P_1 = \emptyset$. We formulate the problem of determining gates as a Dynamic Programming (DP) problem where each activity constitutes a stage. In stage i we need to determine the gate for activity $(i + 1)$. The states for stage i are the gate g_i of activity i and the delay that has already been realized from this gate: $s_i \geq 0$. Let k_i be the difference in time to the next gate. When decision k_i is taken, the next state to be observed is: $(g_{i+1}, s_{i+1}) = (g_i + k_i, (s_i + Y_i - k_i)_+)$ where: $z_+ = \max(0, z)$, and the associated cost during that stage is:

$$p_i[(s_i + Y_i - k_i)_+] + h_i[(k_i - s_i - Y_i)_+].$$

The local income function $\ell[\sigma, \alpha, V(\cdot)]$ of DP (see, e.g., Denardo 1982) expresses the expected cost associated with observing state σ, taking action α and gaining terminal reward $V(\tau)$ if the transition occurs to state τ. It follows that for our problem:

$$\ell_i[(g_i, s_i), k_i, V_{i+1}(\cdot, \cdot)] = p_i E[(s_i + Y_i - k_i)_+]$$
$$+ h_i E[(k_i - s_i - Y_i)_+]$$
$$+ E[V_{i+1}(g_i + k_i, (s_i + Y_i - k_i)_+)].$$

The optimal expected cost from activity i to the end of the project is given by:

$$V_i(g_i, s_i) = \min_{k_i} \ell_i[(g_i, s_i), k_i, V_{i+1}(\cdot, \cdot)].$$

Given that the entire project has a due date d, we can refer to it as a dummy activity $n+1$ that has no duration and for which the optimal expected cost will be:

$$V_{n+1}(g_{n+1}, s_{n+1}) = \begin{cases} 0, & \text{if } g_{n+1} = d, \\ \infty, & \text{if } g_{n+1} \neq d. \end{cases} \tag{1}$$

The optimal expected cost for the entire project is the optimal expected cost from activity 1 to the end of the project $V_1(g_1, 0)$, given that this activity starts at g_1 and has no delays, that is, $s_1 = 0$. The goal of the dynamic program is to find: $\min_{g_1} V_1(g_1, 0)$ (which implies finding optimal values for $g_1, g_2, \ldots, g_n$).

DP employs a process called *recursive fixing* to find the solution to such problems. The term *recursive fixing* means that the optimal values are fixed one after another, in a certain preset sequence (backward or forward). In our case, the determination of any gate, for example gate g_{i+1}, depends on the state (g_i, s_i). The variables s_i can be determined by a forward optimization procedure since we have the initial condition that $s_1 = 0$. In contrast, the variables g_i can be determined by a backward optimization procedure since we have the final condition expressed in (1). In other words, the decision on each gate i depends on the history (gates of activities 1 to $i - 1$) but also on future decisions (gates of activities $i + 1$ to n). Therefore, we cannot use ordinary DP to set gates of all activities in advance.

We are then forced to select a suboptimal approach that will be capable of providing a reasonably good (yet, not necessarily optimal) solution to this problem in reasonable time. We selected the Cross-Entropy approach, which has already proven itself in a variety of other complex contexts, for this purpose. The method and its implementation to our problem are explained in next section.

3 A Cross Entropy approach

3.1 Description of the Cross Entropy method

The Cross Entropy (CE) method, developed by Rubinstein and Kroese (2004), is a general heuristic method for solving estimation and optimization problems. This method has been applied to solve a variety of problems, especially deterministic and stochastic (noisy) combinatorial problems. For example, Alon et al. (2005) presents an application of the CE method to the buffer allocation problem, that is, the problem of allocating buffer spaces amongst machines in a serial production line, so as to optimize some performance measure. The CE

method has also been applied in the field of project management. Cohen et al. (2005) use the method to determine the optimal loading of a stochastic and dynamic multi-project environment by keeping a constant number of projects performed currently in the system and in order to minimize the average makespan of projects. However, the focus of the work of Cohen et al. (2005) is on accepting or blocking projects in a multi-project environment while the work we present here takes the completely different and novel approach of using gates to determine the scheduling of activities.

For optimization problems, a general outline of the CE method requires the translation of the underlying optimization problem into a meaningful estimation problem called the *associated stochastic problem*. Thus, when solving a minimization problem, the estimation problem may be the expected number of times the objective function yields a value which is lower than a pre-specific threshold value. Next, the CE algorithm involves the following two phases: (1) generation of a sample of random data (demands, durations, trajectories, etc.) according to a specified random mechanism, and simultaneous calculation of the objective function values; (2) updating parameters of the random mechanism (on the basis of the data collected) in order to produce an improved sample in the next iteration, that is, a sample that will improve the value of the objective function.

To apply the CE method to our scheduling problem, we first model the case of serial projects. The simple nature of this case allows us to explain better the different steps of the method. We will then extend the application of the CE method to more complex networks, that is, non-serial projects.

3.2 Application of the CE method to serial projects

In addition to the notation provided in Sect. 2.1, we now define:

N	the number of vectors generated at each step
N'	the number of vectors of activity durations generated for each vector of gates
$G = (g_1, \ldots, g_n)$	the vector of gates, where g_k is the gate of activity k
$G_i = (g_{i1}, \ldots, g_{in})$	the ith vector of gates generated
$\hat{P}r_t = (\hat{P}r_{t,1}, \ldots, \hat{P}r_{t,n})$	the distribution vector used to generate a vector of gates in step $t+1$
$\hat{P}r_{t,k,j}$	the probability that the gate of activity k gets the value j in step $t+1$
ρ	the percentage of best results we want to use in each step
$\hat{\gamma}_t$	the sample ρ quantile of the performances (total cost) in step t
$C(G_i)$	the performance of the ith vector generated
$DU(a, b)$	the discrete Uniform distribution defined in the discrete interval $\{a, a+1, \ldots, b\}$, where a and b are the minimal and maximal value of the distribution, respectively
$I_{\{c\}}$	an indicator function which gets a value of 1 if the condition $\{c\}$ is satisfied

First, we describe the different steps of the CE procedure, then we present the algorithm. In the first step, we set the counter to 1 and define the initial distribution of gates. Since we have no *a priori* information on the value of the gates, we could have started from a discrete uniform distribution between 0 and the due date of the project d. Then, the number of possible vectors of gates would be: $(d+1)^n$. In CE, there is a direct relationship between

the number of possibilities and N, the number of vectors we have to generate in each step. Hence, in order to limit N, we have to limit the number of possibilities for the vector of gates. The first way to limit this number is to use the constraint: $g_k \geq g_{k-1}$ for all $k = 2, \ldots, n$. The second way is to fix the gate of activity k such that there exists a chance to finish remaining activities on time. For example, the gate of activity n should not be larger than $d - a_n$, otherwise with probability 1 we will end the last activity after the due date. For activity k, $k = 1, \ldots, n$, the minimal time to complete the remaining activities is: $\sum_{u=k}^{n} a_u$, therefore the gate should not be larger than $d - \sum_{u=k}^{n} a_u$.

In the second step, we generate a sample of N vectors of gates and calculate the cost of the project for each vector. To calculate the cost of a specific vector of gates G_i, we generate N' vectors of activity durations, each activity i according to the probability function of Y_i, then we calculate for each realization the cost of the project. Finally, we calculate the average of these N' costs. This average is the estimator of the expected cost of the project for the specific vector of gates G_i. For a specific realization of activity durations $y = (y_1, \ldots, y_n)$, the cost is: $\sum_{k=1}^{n} (h_k (g_{k+1} - t_k - y_k)_+ + p_k (t_k + y_k - g_{k+1})_+)$ where t_k is the start time of activity k, $t_1 = g_1$, $t_k = \max\{g_k, t_{k-1} + y_{k-1}\}$ and $g_{n+1} = d$. Then, these costs are ordered from the smallest to the largest. The choice for the sample size N depends, according to Rubinstein and Kroese (2004), on the number of parameters we have to estimate. That is, if we have to estimate k parameters, we should use a sample size $N = ck$ where c is a constant (typically, $5 \leq c \leq 10$). As explained above, in our case the possible number of gates is in the order of magnitude of d^n (maybe less according to the limitations applied in the first step). For the small examples of Sect. 4, we used $N = 10,000$. For larger projects, the number cd^n may be astronomical. We will overcome this problem in Sect. 3.4.

In the third step we update the distribution of gates according to results of the previous step. For each good performance, that is, lowest costs, we note the value of the gates. The updated probability for each gate value will be the frequency of this value in the best performances, that is, the number of times this value appeared divided by the number of all best performances. The number of best performances considered is equal to ρN where ρ is suggested to be between 0.01 to 0.2 (see Rubinstein and Kroese 2004). In our examples, we took $\rho = 0.1$, that is, we considered the 10% best performances of the sample.

In the fourth step, after calculating the probability vector according to the frequencies, as explained above, we update it using a smoothing procedure, as explained in Rubinstein and Kroese (2004):

$$\hat{P}r_t = \alpha \hat{P}r_t + (1 - \alpha)\hat{P}r_{t-1}.$$

The reason for using this smoothing procedure is to prevent from the vector of probabilities to converge too fast to values like 0 and 1, since when it happens, it will often remain in the same state and this is undesirable. From Rubinstein and Kroese (2004), α should be between 0.3 to 0.9. In our examples, we used $\alpha = 0.7$.

In the fifth step, we choose one of the possible stopping criteria described in Rubinstein and Kroese (2004). We stop when the updated vector of probabilities has converged to a binary vector, that is, when for each gate, the probability that it is equal to a specific value is close to 1. Otherwise we have to continue for another iteration.

The CE procedure for our specific problem is given in the following algorithm:

Algorithm 3.2 (CE algorithm for serial projects)

1. Set $t = 1$, $\hat{P}r_{0,k} = DU(g_{k-1}, d - \sum_{u=k}^{n} a_u)$ for $k = 2, \ldots, n$ and $\hat{P}r_{0,1} = DU(0, d - \sum_{u=1}^{n} a_u)$.

2. Generate a sample $G_1, \ldots, G_N$ from $\hat{P}r_{t-1}$ and calculate the performance (the total cost of the project) for each vector: $C(G_i)$. Order these performances from smallest to largest and let $\hat{\gamma}_t = C(G_{(\lceil \rho N \rceil)})$.

3. Update $\hat{P}r_t = (\hat{P}r_{t,1}, \ldots, \hat{P}r_{t,n})$ according to:

$$\hat{P}r_{t,k,j} = \frac{\sum_{i=1}^{N} I_{\{C(G_i) \le \hat{\gamma}_t\}} I_{\{g_{ik}=j\}}}{\sum_{i=1}^{N} I_{\{C(G_i) \le \hat{\gamma}_t\}}}. \tag{2}$$

4. Update $\hat{P}r_t$ using the smoothing procedue:

$$\hat{P}r_t = \alpha \hat{P}r_t + (1 - \alpha) \hat{P}r_{t-1}.$$

5. If for some t, for some j and for all k, $\hat{P}r_{t,k,j} > 0.95$, then stop; otherwise set $t = t + 1$ and reiterate from Step 2.

3.3 Extension of the CE method to non-serial projects

3.3.1 Direct extension of the CE method to non-serial projects (DCE)

This extension is based on the same algorithm that was developed in Sect. 3.2 for serial projects. The differences between the serial and non-serial algorithms are in the initial distribution of the vector of gates and in the calculation of the cost.

- The initial distribution of the vector of gates: to limit the number of possibilities for vectors that need to be generated, we can use the constraint adapted to the network structure of the project: $g_k \ge g_j$ for all $j \in P_k$, for all $k = 2, \ldots, n$. In serial projects, the second way to limit this number is to fix the gate of activity k such that there exists a chance to finish the remaining activities on time. As explained above, this meant that the gate could not be larger than $d - \sum_{u=k}^{n} a_u$. In non-serial projects, it is more complicated to know for a specific activity the remaining time until the completion of the project. Using the fact that there is one terminal activity n, we bounded gates of all activities by $d - a_n$.
- The cost calculation: in non-serial projects, for a specific realization of activity durations $y = (y_1, \ldots, y_n)$, the cost is:

$$\sum_{k=1}^{n} \left[\sum_{j \in S_k} (h_k(g_j - t_k - y_k)_+ + p_k(t_k + y_k - g_j)_+) \right]$$

where t_k is the start time of activity k, $t_1 = g_1$, $t_k = \max_{j \in P_k}\{g_k, t_j + y_j\}$ and $g_{n+1} = d$.

3.3.2 A heuristic method based on the serial CE method

When we apply the direct extension of the CE method to non-serial projects with a large number of activities, the number of possible vectors of gates grows. Therefore, the number N of generated vectors should also grow, which increases considerably the running time of the algorithm. To overcome this difficulty, we used the following heuristic method based on the CE method applied on the paths (CEP) of the project:

- Find the critical path of the non-serial project, that is, the path with the largest expected duration.

- Apply the algorithm for serial projects developed in Sect. 3.2 to the critical path and find the gates for its activities.
- For each non-critical path, apply the algorithm for serial projects developed in Sect. 3.2 when the due date of this path will be the gate of the activity in which that path merges with the critical path.

This heuristic algorithm reduces the size of the problem. However, this method has two drawbacks: the network structure is not preserved and the problems solved are still quite small.

3.4 CE method using a continuous distribution

To overcome the difficulties described in Sect. 3.3.2, we use a continuous distribution function for the generation of the vector of gates instead of a discrete distribution function. As explained in Sect. 3.2, the choice for the sample size N depends on the number of parameters we have to estimate. If, as explained in Kroese et al. (2006), we use the Normal distribution function, we will have to estimate, for each gate, just two parameters—the mean and the standard deviation. We will then have a total of $2n$ parameters to estimate. Therefore, even for projects with up to $n = 50$ activities we will have to use a sample size $N \leq 2cn = 1,000$ for $c = 10$.

The additional notation used in this section is as follows:

$\hat{\mu}_t = (\hat{\mu}_{t,1}, \ldots, \hat{\mu}_{t,n})$ the mean of the normal distribution used to generate a vector of gates in step $t + 1$

$\hat{\sigma}_t^2 = (\hat{\sigma}_{t,1}^2, \ldots, \hat{\sigma}_{t,n}^2)$ the variance of the normal distribution used to generate a vector of gates in step $t + 1$

$\mathcal{N}^{elite}$ the set of indexes of the $\lceil \rho N \rceil$ vectors that give the best performances among the N vectors generated

C_t^* the best performance (lower cost) obtained in step t

$NLIM$ the maximum number of variance injections to perform in the algorithm

Again, we start by describing the different steps of the CE procedure using a continuous distribution and then we present the algorithm. In the first step, we choose the initial mean and variance. In our case, we chose for each activity its early start time as the initial mean and $\hat{\sigma}_0^2 = (\frac{d^2}{9}, \ldots, \frac{d^2}{9})$ as the initial variance (that is, $\hat{\sigma}_{0,j} = d/3$ for all $j = 1, \ldots, n$). This initial value for the variance ensures that when the procedure starts it is possible to generate gates from all the interval $[0, d]$. Indeed, in a Normal distribution, 99.7% of the values lie within three standard deviations of either side of the mean. With our choice of $\hat{\sigma}_{0,j} = d/3$ for all $j = 1, \ldots, n$, 99.7% of the values generated will be between the mean (the early start time of the activity) plus d and the mean minus d.

In the second step, we generate N vector of gates from the Normal distribution with the initial mean and standard deviation. For each vector of gates generated we calculate the cost of the project. We then choose the $\lceil \rho N \rceil$ vectors that give the best performances and keep the indexes of these vectors in the set $\mathcal{N}^{elite}$, where the cardinality of the set $\mathcal{N}^{elite}$ is equal to $\lceil \rho N \rceil$: $|\mathcal{N}^{elite}| = \lceil \rho N \rceil$. For example, for $\rho = 10\%$ and for $N = 1,000$, the set $\mathcal{N}^{elite}$ will be composed of indexes of the 100 vectors of gates that gave lower costs among the $N = 1,000$ vectors generated.

In the next step we update for each activity just two parameters: its mean and standard deviation (instead of updating probability values for each one of the possibilities for this gate as was done in Sect. 3.3.2). In the next iteration, the mean of the Normal distribution

will be the average of the gates that gave the best performances (3) and the variance will be the variance of these gates (4).

In the fourth step, we use a smoothing procedure as explained in Sect. 3.2. For the smoothing of $\hat{\mu}_t$ and $\hat{\sigma}_t^2$ we use $\alpha = 0.7$. With this simple smoothing, the convergence to a degenerate distribution may happen too quickly and may lead to a sub-optimal solution. One possibility to overcome this problem is to follow the suggestions of Kroese et al. (2006) and use dynamic smoothing for the smoothing of $\hat{\sigma}_t^2$. For this purpose, (5) is replaced by the following one: $\hat{\sigma}_t^2 := \beta_t \hat{\sigma}_t^2 + (1 - \beta_t)\hat{\sigma}_{t-1}^2$ where: $\beta_t = \beta - \beta(1 - \frac{1}{t})^q$, q is an integer typically between 5 and 10 and β is a smoothing constant typically between 0.8 and 0.99. The problem in this method is that now the algorithm has a very slow convergence, which can make difficult the choice of a good stopping criterion. The second possibility is to follow the suggestion of Botev and Kroese (2004) and use the technique of "variance injection": if at iteration t the maximal variance in the vector $\hat{\sigma}_t^2$ is lower than ε (we use $\varepsilon = 0.01$), add to all the variances of the vector the value $|C_t^* - C_{t-1}^*|h$ for some h between 0.1 and 10 (we use $h = 2$).

In step 5, we have to stop if the stopping criterion is met. Botev and Kroese (2004) propose the following stopping criterion: stop when the number of variance injections exceeds a specific limit *NLIM* (we use *NLIM* = 5).

Of course, the gates obtained from applying this algorithm are continuous values, but our solution is restricted to discrete values. Rather than using arbitrary rounding, we apply the CE algorithm again using the discrete distribution (developed in Sect. 3.3.1) by generating a vector of gates according to the Uniform discrete distribution: $DU(G^F, G^C)$ where G^F and G^C are vectors composed of the "floor" and "ceiling" values, respectively, of the continuous vector of gates. Since we have two possibilities for each gate, the number of vector to be generated N has to be proportional to $2n$. For large projects (up to 50 activities), $N = 1,000$ is large enough and will ensure a fast convergence.

The CE procedure using a continuous density function is given in the following algorithm:

Algorithm 3.4 (CE algorithm using a continuous density function)

1. Set $t = 1$, choose $\hat{\mu}_0 = (\hat{\mu}_{0,1}, \ldots, \hat{\mu}_{0,n})$ and $\hat{\sigma}_0^2 = (\hat{\sigma}_{0,1}^2, \ldots, \hat{\sigma}_{0,n}^2)$.
2. Generate a random sample $G_1, \ldots, G_N$ from the Normal distribution $N(\hat{\mu}_{t-1}, \hat{\sigma}_{t-1}^2)$ and calculate the performance (the total cost of the project) for each generated vector: $C(G_i), i = 1, \ldots, N$. Order these performances from smallest to largest and let $\mathcal{N}^{elite}$ be the set of indices of the $\lceil \rho N \rceil$ vectors that give the best performances among the N vectors generated.
3. Update for all $j = 1, \ldots, n$:

$$\hat{\mu}_{tj} := \sum_{i \in \mathcal{N}^{elite}} g_{ij}/|\mathcal{N}^{elite}| \tag{3}$$

and

$$\hat{\sigma}_{tj}^2 := \sum_{i \in \mathcal{N}^{elite}} (g_{ij} - \hat{\mu}_{tj})^2/|\mathcal{N}^{elite}|. \tag{4}$$

4. Smooth:

$$\hat{\mu}_t := \alpha \hat{\mu}_t + (1 - \alpha)\hat{\mu}_{t-1},$$
$$\hat{\sigma}_t^2 := \alpha \hat{\sigma}_t^2 + (1 - \alpha)\hat{\sigma}_{t-1}^2. \tag{5}$$

If $\max_{j=1,\ldots,n} \hat{\sigma}_{tj}^2 \leq \varepsilon$, $\hat{\sigma}_t^2 := \hat{\sigma}_t^2 + |C_t^* - C_{t-1}^*|h$.

5. If the stopping criterion is met, then stop; otherwise set $t = t + 1$ and reiterate from Step 2.

4 Computational study

The concept of setting gates for project activities is relatively new and, at least as far as we know, no one before us has proposed a specific method to determine such gates. Therefore, to study the performance of the algorithms developed in Sect. 3, we compare each of them to four other heuristic methods:

- Early Start (ES). In this method, gates are determined according to their early start time with respect to time 0 and with no regard to costs. Therefore, $g_1 = 0$ and $g_k = \max_{j \in P_k} \{g_j + E(Y_j)\}$ for all $k = 2, \ldots, n$.
- Late Start (LS). In this method, gates are determined according to their late start time with respect to the due date of the project and again, with no regard to costs. Therefore, $g_{n+1} = d$ and $g_k = \min_{j \in S_k} \{g_j - E(Y_k)\}$ for all $k = 1, \ldots, n$.
- Random Gates (RG). In this method, we randomly generate vectors of gates as follows: for each activity, we generate a gate uniformly distributed between its early start time and its late start time. For each vector of gates generated, we calculate the total cost of the project. We run this procedure (generation of vectors and calculation of cost) using the same CPU time that the CE algorithm took to converge, and choose the vector of gates that yields the lowest cost.
- Simulated Annealing (SA). SA is a general purpose heuristic that has been used to solve various combinatorial problems. A detailed description of SA can be found in Heragu (1997) or Golany et al. (1999). The SA heuristic starts with an initial feasible solution and an initial "temperature". Then, it randomly generates a neighbor solution. If the new solution has a better performance, it is accepted. Otherwise, it can still be accepted with a probability that depends on the degradation in the objective function and on the temperature of the process at that stage. This routine is repeated until some termination conditions are met. Then, the temperature is decreased and the same steps are repeated. The detailed SA algorithm which was adapted and used in the present paper is presented in Appendix A.

4.1 Application of the CE algorithm with discrete distributions

As explained above, the discrete CE procedure can be used to solve only small examples, that is, with up to 15 activities. This can help us to get some first insights on the performance of the algorithms developed in this paper. Larger examples are solved in the next section (Sect. 4.2). The first step in the computational study is to generate projects on which we will apply the CE and the heuristics to which it is compared.

4.1.1 Generation of a serial project

For each activity i, we need to generate the following parameters: a_i, b_i, h_i, p_i. These parameters were uniformly generated in an interval between some minimal and maximal values. This allows many possible configurations for a project that could be composed of short or long activities, cheap or expensive penalties. One possible configuration for small projects could be: $a_i \sim DU[2, 10]$, $b_i \sim DU[8, 20]$, $h_i \sim DU[1, 5]$, $p_i \sim DU[3, 7]$. The penalty p_n for the last activity should be significantly larger than the penalty of the other activities since

it represents the penalty for not finishing the project on the due date. For small projects, we set $p_n = 4p_{max}$ where p_{max} is the maximal value of the interval wherein the penalty p_i was generated. In addition, for the entire project we need to generate a due date. To do so, we calculate what we called d_{min}, which is the minimal finishing time of the project. In a serial project, d_{min} is simply the sum for all activities at the minimal processing times: $d_{min} = \sum_{i=1}^{n} a_i$. In the same way, we calculate d_{max}, which is the maximal finishing time of the project. In a serial project, d_{max} is simply the sum for all activities at the maximal processing times: $d_{max} = \sum_{i=1}^{n} b_i$. Then, we generate d uniformly in the interval $[d_{min}, d_{max}]$.

4.1.2 Application of the CE algorithm for serial projects and comparison to other heuristic methods

We programmed the algorithm developed in Sect. 3.2 in Matlab and applied it to a random project with seven activities in series. The data used for the example was randomly generated as explained above and is presented in Table 1.

For this example the due date was set to 85. The gates and the corresponding expected costs obtained in each method are presented in Table 2.

We observe in Table 2 that for the due date $d = 85$, the CE method gave the best results. The gates and the cost obtained by the RG method were close to those of the CE method. However, for the ES (LS) method, only the gates of earlier (later) activities are close to that of CE method. The performance of the ES method can be explained by observing that the due date was rather close to the expected duration of the project and that the gates in the ES method are determined as the early start time of each activity according to the expected durations. To check the influence of postponing the due date, we used the same example presented in Table 1 but with a due date $d = 100$. Of course, the gates obtained in the ES method were not changed since they do not depend on the due date. The gates and the corresponding expected costs obtained in each method are presented in Table 3.

Now, the advantage of the CE method becomes obvious. While the CE method responds to the change in data parameters by adjusting gates so as to minimize the expected cost, ES and LS methods can not respond to the change and continue to determine gates according to durations only. There are many interesting facts that we can see from this example. One of them is that the cost of the LS method remains unchanged. The LS method sets gates

Table 1 The data for a project with seven activities in series

Activity	1	2	3	4	5	6	7
a_i	8	4	3	2	9	4	4
b_i	20	8	19	10	15	18	10
h_i	2	2	4	3	1	1	3
p_i	4	6	6	4	3	3	28

Table 2 The results for a project with seven activities in series with $d = 85$

	g_1	g_2	g_3	g_4	g_5	g_6	g_7	Mean cost
ES	0	14	20	31	37	49	60	133.91
LS	18	32	38	49	55	67	78	241.88
RG	1	19	25	42	48	62	72	87.61
SA	0	13	25	39	39	52	76	117.29
CE	0	15	25	40	43	62	76	81.86

Table 3 The results for a project with seven activities in a series with $d = 100$

	g_1	g_2	g_3	g_4	g_5	g_6	g_7	Mean cost
ES	0	14	20	31	37	49	60	177.44
LS	33	47	53	64	70	82	93	241.53
RG	10	25	34	46	47	71	92	95.13
SA	0	13	21	38	46	75	92	93.11
CE	13	30	38	53	59	76	92	73.14

Table 4 The data for the non-serial project with seven activities and two paths

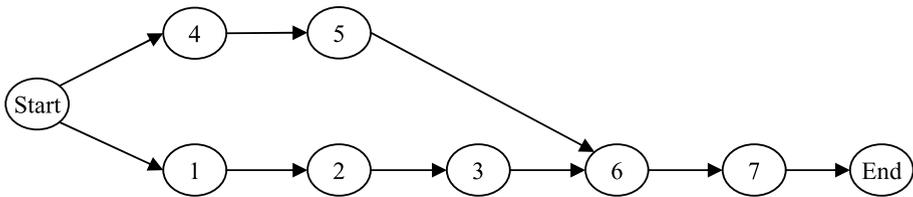| Activity	1	2	3	4	5	6	7
a_i	5	7	5	10	4	8	2
b_i	10	15	8	15	10	10	8
h_i	0.5	2	0.5	1	1	1.5	2.5
p_i	2	5	1	4	3	4	15

Fig. 1 Non-serial project with seven activities and two paths

according to the due date. When the project's due date is changed, the gates are also changed but are still with the same interval between them and with the due date. It is as though we took all the gates and the due date and shifted them forward in time. Note that this would not happen if we use a discount factor since if the due date is shifted to a later time, the gates but also the penalty costs will be shifted to a later time, which will result in a lower cost than the case where the due date is earlier. In contrast to the LS method, in the ES method, where gates are set according to the early start time of activities, the gates remained unchanged but the cost grows (since the penalty due to holding costs grows).

4.1.3 Application of the CE algorithm for non-serial projects and comparison to other heuristic methods

To carry out this comparison, we first needed to generate non-serial projects. This was done by employing the Progen software (see Kolish et al. 1995) that gave us a random number of activities and a random network structure represented by a matrix of successors. Using this matrix we generated the corresponding matrix of predecessors. In addition, for each activity i, we generated the parameters: a_i, b_i, h_i, p_i and d for the entire project as explained in Sect. 4.1.1. Then, we compared performances of the ES, LS RG and SA heuristic methods to the direct extension of the CE method (DCE, developed in Sect. 3.3.1) and to the heuristic method (CEP, developed in Sect. 3.3.2). The first example of non-serial project is presented in Fig. 1. This project includes seven activities and two paths.

The data used for this example is presented in Table 4. For this example the due date was 42. For the CEP method, we first apply the CE method on the serial project 1–2–3–6–7

Table 5 The results for the non-serial project with seven activities and two paths

	g_1	g_2	g_3	g_4	g_5	g_6	g_7	Mean cost
ES	0	7	18	0	12	24	33	47.29
LS	4	11	22	9	21	28	37	85.56
RG	0	7	18	1	15	26	33	46.15
SA	0	7	21	0	14	26	40	73.01
DCE	0	7	18	2	16	26	35	37.59
CEP	0	7	19	3	17	26	36	36.25

Table 6 The data for the non-serial project with ten activities and four paths

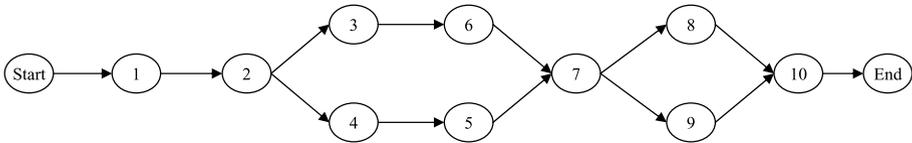| Activity	1	2	3	4	5	6	7	8	9	10
a_i	5	7	10	2	12	4	8	2	4	2
b_i	9	15	16	8	16	10	10	6	6	6
h_i	0.5	2.5	1	2	1.5	1	1.5	0.5	2	2.5
p_i	3	6	2	4	5	3	4	2	4	25

Fig. 2 Non-serial project with ten activities and four paths

with $d = 42$, then we apply the CE method on the serial project 4–5 with $d = g_6$. The gates and the corresponding expected costs obtained in each method are presented in Table 5.

The second example of non-serial project is presented in Fig. 2. This project includes ten activities and four paths.

The data used for this example is presented in Table 6. For this example the due date was set to 62. For the CEP method, we first apply the CE method on the path 1–2–3–6–7–9–10 with $d = 62$. This path is considered as the critical path since its expected duration is the larger. Then we apply the CE method on the path that includes activity 8 with $d = g_{10}$ and on the path 4–5 with $d = g_7$. The gates and the corresponding expected costs obtained in each method are presented in Table 7. In the example presented in Fig. 2 and in Table 6, we can see from results presented in Table 7 that the CE method gave better results than the ES, LS and SA heuristic methods. In addition, we can see a significant advantage of the CEP versus the DCE method. This can be explained by the fact that for the same number of generated vectors, the size of the problem is smaller in the CEP method and this allows the CE method to converge into vector of gates that offer lower costs.

4.2 Application of the CE algorithm with continuous distributions

4.2.1 Application of the continuous CE algorithm on small projects

We first applied the continuous CE algorithm to the examples presented in Sect. 4.1. In the example presented in Table 1 with $d = 85$, the cost obtained by the continuous CE was 67.03, and with $d = 100$, the cost was 71.15. In the example presented in Table 4, the cost

Table 7 The results for the non-serial project with ten activities and four paths

	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}	Mean cost
ES	0	7	18	18	23	31	38	47	47	52	110.23
LS	6	13	24	25	30	37	44	54	53	58	180.4
RG	0	7	20	20	25	34	40	48	48	54	97.94
SA	0	5	19	17	26	23	40	51	45	50	149.05
DCE	0	7	18	22	27	38	44	49	54	58	100.75
CEP	0	6	19	19	25	32	41	50	51	56	66.32

Table 8 Average cost for large projects

		Project size	20–30	30–40	40–50
Average	CE		4104.09	3664.31	5780.84
	ES		5228.82	6360.96	10150.91
	LS		6125.07	6528.80	8113.47
	RG		4809.26	5793.00	8232.54
	SA		5208.11	6046.89	7954.23
Maximal	ES–CE		1654.67	3069.99	7890.89
difference	RG–CE		1082.97	2793.72	3474.42
	SA–CE		1523.08	2942.78	3633.84
Minimal	ES–CE		547.317	2100.01	2785.25
difference	RG–CE		205.70	945.46	946.44
	SA–CE		629.74	1416.23	1716.37

obtained by the continuous CE was 33.45 and for the example presented in Table 6, the cost was 61.54. Thus, in all the examples the continuous CE algorithm led to improved objective function values. This is due to the fact that for the same number of vectors generated, we have less parameters to estimate in the continuous CE, as explained in Sect. 3.4.

4.2.2 Application of the continuous CE algorithm on large projects

As before, we generated networks of projects using Progen (see Kolish et al. 1995). Then, for each activity i, we generated the parameters: a_i, b_i, h_i, p_i and d for the entire project. Since we deal here with large projects, we added a parameter called the type of the activity. For each activity we randomly set a "time type" that determines if it is a long, medium or short activity. Then, for each type we generate the parameters a_i and b_i from different intervals. In the same way, for each activity we randomly set a "cost type" that determines if it is an expensive, medium or inexpensive activity and then, for each type we generate the penalty costs h_i and p_i from different intervals. We programmed the algorithm developed in Sect. 3.4 in Matlab and applied it to five random projects with a random number of activities between 20 to 30, five random projects with a random number of activities between 30 to 40 and five random projects with a random number of activities between 40 to 50. We compared performances of the continuous CE algorithm to the ES, LS RG and SA heuristic methods. The results we obtained are summarized in Table 8.

The results presented in Table 8, clearly indicate the advantage of the continuous distribution CE algorithm over its competitors. In all the 15 project instances that were checked,

the CE method gave the best results. In addition, the differences between the CE method to other methods increase considerably as the size of the project increases. This demonstrates that as the project size grows and more penalty costs are involved, the need to set gates according to a method that take these costs into account grows as well.It is also important to note that the CE algorithms which are presented here are quite robust in the sense that if we apply one of these algorithms several times to the same numerical example, we obtain the same cost and also the same vector of gates.

5 Concluding remarks

This paper solves the stochastic project scheduling problem through a gating approach. The objective is to determine the gate for each activity in order to minimize expected holding and shortage costs. To achieve this, we chose the Cross-Entropy method. We first applied the method to serial and non-serial projects using a discrete distribution in small examples. Then, we applied the CE method using a continuous distribution on larger examples. To check the performance of the algorithms developed, we compared them to four heuristic methods: the Early Start (ES) heuristic method, the Late Start (LS) heuristic method, a random method (RG) and the Simulated Annealing (SA) heuristic method. In all examples, the CE-based algorithms developed in this paper outperformed other methods to which they were compared. Furthermore, these CE algorithms were found to be robust. The advantages of the algorithms developed grows as the size of the project grows since more penalty costs are involved and also because the uncertainty in the start times of activities grows as well. This strengthens the need to set gates according to a method that takes into account the holding and shortage costs in addition of the duration parameters which are used by simpler methods.

Appendix A: Specification of the SA algorithm

A.1 Notation

The additional notation used in this section is as follows:

ρ the percentage by which the gate of an activity changes to create a new solution
TLimit   the number of times the temperature is decreased
T_r the temperature used in step r, $r = 1, \ldots, TLimit$
$N_{\max}$ maximum number of solutions evaluated at each temperature
δ the difference between the cost of the candidate solution and the cost of the current solution
CR   the cooling rate by which the temperature is decreased

A.2 Generation of neighboring solutions

1. Randomly select a number i between 1 to n.
2. Randomly select sign to be equal to 1 or -1.
3. Set: $g_i = g_i + \mathrm{sign}(\rho g_i)$ where ρ was set to 0.1.
4. Check feasibility of g_i: if $g_i < 0$ or $g_i > d$, goto 1, otherwise stop.

A.3 Simulated annealing algorithm

1. Set $r = 1$, $T_r = 0.1$.
2. Select as an initial solution for the gates the vector composed of the early start times of the activities, and compute the cost of the initial solution.
3. Repeat steps 4–5 N_{max} times, where N_{max} was set to 1,000.
4. Select a neighbor solution according to Appendix A.2 and compute its cost.
5. If the cost of the candidate solution is lower than the current cost, accept the candidate solution with probability 1. Otherwise, accept it with probability $e^{-\delta/T_r}$ where δ is the difference between the cost of the candidate solution and the cost of the current solution ($\delta > 0$).
6. Set $r = r + 1$. If $r \leq TLimit$ (*TLimit* was set to 10), set $T_r = T_{r-1}CR^{r-1}$, where CR was set to 0.9 and goto 3. Otherwise, stop.

References

Alon, G., Kroese, D. P., Raviv, T., & Rubinstein, R. Y. (2005). Application of the Cross-Entropy method to the buffer allocation problem in a simulation-based environment. *Annals of Operations Research*, *134*, 137–151.

Botev, Z., & Kroese, D. P. (2004). Global likelihood optimization via the cross-entropy method with an application to mixture models. In R. G. Ingalls, M. D. Rossetti, J. S. Smith, & B. A. Peters (Eds.), *Proceedings of the 2004 winter simulation conference* (pp. 529–535). New York: IEEE Press.

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

Cohen, I., Golany, B., & Shtub, A. (2005). Managing stochastic, finite capacity, multi-project systems through the Cross-Entropy methodology. *Annals of Operations Research*, *134*, 189–199.

Demeulemeester, E. L., & Herroelen, W. S. (2002). *Project scheduling—a research handbook*. Dordrecht: Kluwer Academic.

Denardo, E. V. (1982). *Dynamic programming: models and applications*. New York: Prentice-Hall.

Elmaghraby, S. E. (1977). *Activity networks: project planning and control by network models*. New York: Wiley.

Elmaghraby, S. E. (2001). On the optimal release time of jobs with random processing times, with extensions to other criteria. *International Journal of Production Economics*, *74*, 103–113.

Elmaghraby, S. E., Ferreira, A. A., & Tavares, L. V. (2000). Optimal start times under stochastic activity durations. *International Journal of Production Economics*, *64*, 153–164.

Fazar, W. (1959). Program evaluation and review technique. *American Statistician*, *13*(2), 10.

Golany, B., Dar-El, E. M., & Zeev, N. (1999). Controlling shop floor operations in a multi-family, multi-cell manufacturing environment through constant work-in-process. *IIE Transactions*, *31*, 771–781.

Goldratt, E. M. (1997). *Critical chain*. New York: North River Press.

Heragu, S. (1997). *Facilities design*. Boston: PWS Publishing Company.

Herroelen, W. S., & Leus, R. (2005). Project scheduling under uncertainty: survey and research potentials. *European Journal of Operational Research*, *165*, 289–306.

Herroelen, W. S., Van Dommelen, P., & Demeulemeester, E. L. (1997). Project network models with discounted cash flows: a guided tour through recent developments. *European Journal of Operational Research*, *100*, 97–121.

Herroelen, W. S., De-Reyck, B., & Demeulemeester, E. L. (1998). Resource-constrained project scheduling: a survey of recent developments. *Computers and Operations Research*, *25*(4), 279–302.

Kimms, A. (2001). *Mathematical programming and financial objectives for scheduling projects*. Dordrecht: Kluwer Academic.

Kolish, R., & Padman, R. (2001). An integrated survey of deterministic project scheduling. *Omega*, *29*, 249–272.

Kolish, R., Sprecher, A., & Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, *41*(10), 1693–1703.

Krishnamoorthy, M. (1968). Critical path method: a review. Technical Report, Michigan Univ., Dept. of Industrial Engineering, Ann-Arbor, MI, USA.

Kroese, D. P., Rubinstein, R. Y., & Porotsky, S. (2006). The cross-entropy method for continuous multi-extremal optimization. *Methodology and Computing in Applied Probability*, *8*, 383–407.

Malcom, D. G., Roseboom, J. H., Clark, C. E., & Fazar, W. (1959). Apllication of a technique for research and development of program evaluation. *Operations Research*, *7*, 646–669.

Rand, G. K. (2000). Critical chain: the theory of constraints applied to project management. *International Journal of Project Management*, *18*, 173–177.

Rubinstein, R. Y., & Kroese, D. P. (2004). *The Cross-Entropy method—a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*. New York: Springer.

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

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*(3), 697–705.

Trietsch, D. (2006a). Economically balanced criticalities for robust project scheduling and control. Working Paper, ISOM Department, University of Auckland, New Zealand.

Trietsch, D. (2006b). Optimal feeding buffers for projects or batch supply chains by an exact generalization of the newsvendor result. *International Journal of Production Research*, *44*, 627–637.