



## Managing Stochastic, Finite Capacity, Multi-Project Systems through the Cross-Entropy Methodology\*

IZACK COHEN

BOAZ GOLANY

AVRAHAM SHTUB<sup>†</sup>

shtub@ie.technion.ac.il

Industrial Engineering and Management, Technion—Israel Institute of Technology, Haifa, Israel 32000

Received January 2003; Revised August 2003; Accepted November 2003

**Abstract.** This paper addresses the problem of loading a finite capacity, *stochastic (random)* and *dynamic multi-project system*. The system is controlled by *keeping a constant number of projects concurrently in the system*. A new approach, based on the Cross-Entropy (CE) method, is proposed to determine optimal loading of the system. Through numerical experiments, we demonstrate the CE method performance and show new insights into its behavior in a noisy system. Particularly, we suggest a trade-off between the convergence time, the number of iterations and the noise level.

**Keywords:** project management, stochastic processes, uncertainty modeling, simulation, finite capacity, cross-entropy

Many organizations perform several projects concurrently utilizing the same pool of scarce resources. These *multi-project organizations* are under constant pressure to improve their performance due to global competition and challenging business realities, which lead to shorter project duration, increasing project complexities, fewer resources and a need to employ newer technologies. Consequently, *multi-project organizations* are seeking improved project management techniques. Traditional project management techniques (e.g., PERT/CPM) do not fully address the finite capacity, *stochastic (random)* and *dynamic* nature of *multi-project systems* (e.g., Abdel-Hamid and Madnick, 1991; Archibald and Villoria, 1967; Davis, 1974). In recent years, a number of new methodologies have been developed with the specific purpose of accounting for the complex characteristics of *multi-project organizations*. Critical Chain (CC) Goldratt (1997), Constant number of Projects In Process (CONPIP) and Constant Time In Process (CONTIP) (Anavi-Isakow and Golany, 2003) are examples of such methodologies.

All three methodologies were motivated by concepts originally developed to control production systems. They combine *scheduling* with *finite-capacity loading* procedures. Projects are loaded into a system subject to capacity constraints. Most multi-project research has focused on scheduling (see typical examples in Kurtulus and Davis, 1982;

\*This research was partially supported by the Inga and Hal Marcus Research Fund.

<sup>†</sup>Corresponding author.

Sprenza and Vercellis, 1993) while relatively minor attention has been devoted to issues that relate to *optimal loading policy*. The existing literature (Anavi-Isakow and Golany, 2003; Herroelen and Leus, 2001; Leach, 1999), suggests “rules of thumb” as a basis for loading decisions. None of the earlier techniques is capable of producing optimal performance and, in the few cases that were exhaustively explored through full enumeration, it was revealed that the rules of thumb might yield rather poor performance (e.g., Herroelen and Leus, 2001).

In this paper, we propose a Cross-Entropy (CE) based method (Rubinstein, 1999) to determine optimal loading of multi-project systems. We demonstrate the approach through CONPIP—a methodology that manages the load in the system by limiting the maximal number of projects in process (*NPIP*). The proposed methodology models *congestion effects* that might be present in the system by implementing penalties for waiting in queues through monotonically increasing functions.

Consider an environment of multiple concurrent non-unique projects. The projects share common characteristics (e.g. similar precedence relations among their activities), and they compete for the same set of scarce resources. For example, such an environment is typical in organizations doing maintenance or retrofit projects in the aircraft industries. In this industry it is quite common to find plants working on 3–5 aircraft types, with 5–10 aircraft in each type. These numbers are typically rendered by capacity constraints where the most important limiting factor is the available hangar area. Following Adler et al. (1995), we use a model based on stochastic *processing networks*. The building blocks of the model are interdependent resources that process project activities. At any given moment, each activity is either receiving service from a resource, queuing up for access to a resource (in a *resource queue*) or waiting to join a predecessor activity that is being processed or delayed elsewhere (in a *synchronization queue*). The network model is stochastic (that is, activities’ durations are modeled by random variables) and dynamic (that is, as time goes by new projects randomly appear and existing projects are completed). Randomness here captures unpredictable variability (both in the arrival pattern of new projects and in their required processing times) that is prevalent and significant in most multi-project environments.

The rest of the paper is organized as follows: In the next section, we discuss some of the management methodologies for stochastic, dynamic, multi-project systems. Section 2 includes the development of the CE algorithms for CONPIP and in Section 3 we demonstrate CE performance via two numerical examples. Section 4 presents our concluding remarks.

## 1. Managing multi-project systems

The majority of the literature on multi-project management assumes static, deterministic environments (e.g., Kurtulus and Davis, 1982; Sprenza and Vercellis, 1993). For the most part, research efforts were directed at solving a scheduling problem known in the literature as the *multi-project resource constrained scheduling problem (MPRCSP)*.

In contrast, the subject of finding the optimal loading point of a finite-capacity stochastic, dynamic, multi-project system has attracted little attention.

We start this section with a short review of finite-capacity loading in production systems. Then, we continue and describe three multi-project management methodologies, CC, CONTIP and CONPIP.

### 1.1. Finite-capacity loading of production systems

In some finite-capacity loading models (Kekre, 1987; Banker, Datar, and Kekre, 1988; Karmarkar and Kekre, 1992; Yang and Deane, 1993; Matsuura, Tsubone, and Kanezashi 1996; Tielemans and Kuik, 1996), a production facility is modeled as a *queuing system* where the expected *queuing delay* serves as a performance measure. For a high level of utilization (i.e., high system load), the expected queuing delay dominates the expected total manufacturing *stay-time* (defined as queuing delay plus processing time). This outcome is consistent with known results from *queuing theory* and can be further explained through *congestion curves*, see, e.g. Adler et al. (1996). An example of this approach can be found in Kuik and Tielemans (2003) who use the average stay-time in the system as a performance measure. They consider a single-machine system with setup times and provide analytical approximations for optimal batch sizes that minimize the stay-time in the system. However, the development of an analytical model for complicated queuing systems has remained intractable and the common experimental tool is therefore simulation (see Law and Kelton, 1991, pp. 114–115).

### 1.2. Finite capacity in multi-project environments

Multi-project management methodologies use a set of parameters to control the load in the system (*loading parameters*). To define loading parameters for a multi-project system we assume that the following prerequisites hold:

- (1) One can assign a *numerical value* to each loading parameter and these values can be characterized by a *discrete state vector*  $x \in X$ , where  $X$  is a finite set of states.
- (2) The particular performance measure denoted as  $S(x)$  (the *performance function*), is a real function of  $x$ ,  $\forall x \in X$ .
- (3) Appropriate selection of  $x$  (say  $x^*$ ) yields optimal system performance (i.e., produce optimal or near optimal solutions,  $S(x^*)$ ).

Next, we discuss three control methodologies that were suggested in the context of multi-project systems, paying particular attention to the question of how and to what extent these methods determine good values for their respective loading parameters.

### 1.2.1. *CC methodology for multi-project management and control*

The CC methodology, introduced by Goldratt (1997), aims at developing a sound schedule, using *buffer management*, in order to avoid schedule overruns. A buffer stores the time by which the project activities corresponding to this buffer could be delayed without causing a delay to the planned project due-date. CC generates a schedule by solving a deterministic MPRCSP and subsequently protects the schedule from uncertainty through the insertion of buffers. Buffer management amounts to dynamic management of resources according to buffer consumption levels. Amongst several competing activities, top priority in resource allocation is given to the activity whose relative buffer consumption is the highest.

CC implements finite-capacity loading according to the following heuristic approach, as illustrated by Leach (1999): First, identify the *bottleneck resource*, namely the most constraining resource (often based on managerial experience). Then, stagger the release of projects into the system so that the bottleneck has no idle time. Next, associate a time buffer, called the capacity buffer, with the bottleneck. The capacity buffer role is to ensure bottleneck availability; it decouples between bottleneck activities that belong to successive projects, thus determining the planned projects' start-times. However, there is no standard way in the CC literature to set release rate or to determine the size of the capacity buffer, meaning that the methodology did not establish an approach to determine the load in the multi-project system. In some of the CC literature (e.g., Herroelen and Leus, 2001; Leach, 1999), the methodology is to add another buffer type in front of the bottleneck, namely a *drum buffer*. This buffer amounts to keeping a constant amount of work in front of the bottleneck resource to minimize starvation. Again, CC does not provide quantitative models for setting the capacity of the drum buffer. It is no surprise then, that CC does not address the optimality level of different values of the loading parameters.

### 1.2.2. *CONTIP and CONPIP methodologies for multi-project management and control*

The practice of beginning the work on an incoming project immediately upon its arrival is equivalent to the “*push*” approach in production management. We consider a “*pull*” approach that allows new work to enter only when the production system signals that it can accept it. Anavi-Isakow and Golany (2003) applied the “*pull*” principle using the CONWIP method (see Hopp and Spearman, 1996) in a dynamic, stochastic, multi-project system and showed (by simulation) that its performance was superior over a push-based system. (Specifically, the average total stay-time of a project—from its arrival into the system until its exit—was lower using the CONPIP or CONTIP methodologies).

CONTIP (constant time in process) controls the amount of processing time required by all the projects that are in the system. New projects are allowed into the system only when the requirement for processing time drops below a predetermined upper bound (*TPIP*). Otherwise, projects wait in a backlog queue. Each time an activity ends, the remaining requirement for processing time is updated. To be processed, an activity has to be ready (i.e., all its predecessor activities have been completed) and the relevant resource has to be available. Thus, the completion time of an activated project is dependent upon the status of the system it meets upon its arrival.

CONPIP limits the number of projects that are allowed in the system to a fixed number ( $NPIP$ ). If a project arrives when the number of projects in the system equals  $NPIP$ , it will wait in a backlog queue. When a project is completed, the first project in the backlog queue enters the system. Otherwise, if there are less than  $NPIP$  projects, the backlog queue is empty and incoming projects are immediately admitted into the system. Once started, a project is broken down to its individual activities. As in CONTIP, to be processed, an activity has to be ready (i.e., all its predecessor activities have had to be completed) and the relevant resource has to be available. Thus, the completion time of an activated project is dependent upon the status of the system it meets upon its arrival.

In both CONPIP and CONTIP, activities are penalized for long delays in resource queues through monotonically increasing functions. The penalties reflect real life corrective or updating actions that are needed before the delayed activity is done (e.g. in aircraft maintenance projects, an engineer usually needs to update the data of a delayed activity, to repeat some inspections, etc.).

Through setting the optimal value of  $NPIP$  or  $TPIP$  the system may achieve better performance. However, there is no established way to determine the optimal value of the loading parameters,  $TPIP$  and  $NPIP$ , respectively. Anavi-Isakow and Golany (2003) ran a series of simulation experiments, and used the performance function values to determine the different loading parameter values.

This paper focuses on CONPIP as a management methodology for finite capacity, non-unique, multi-project systems, such as maintenance lines in the aircraft industry (both civilian and military). In such organizations, there is usually *some flexibility* in altering the preplanned project start-time without incurring costs to the customer, who continues to utilize the aircraft. This is especially true, when the organization is doing fleet maintenance for a customer. In that situation, usually, the customer cannot afford to release an additional aircraft (beyond a certain threshold) to maintenance until one aircraft returns to service.

Using the CE approach, we select the number of projects in the system to minimize the average total stay-time in the system. Our approach is particularly important for scenarios in which the range for loading parameter values, as suggested by Anavi-Isakow and Golany (2003), might be perceived as too “wide” by the managers of the multi-project system or when there is a need to determine  $NPIP$  values for several project types (in which case we get a “noisy” combinatorial optimization problem).

It is worth noting that Anavi-Isakow and Golany (2003) further improve the performance of CONPIP by applying more sophisticated priority rules to manage the queues in the system, rather than the standard FCFS rules that we use here. However, this paper focuses on the selection of the loading parameter.

## 2. CE algorithm for $NPIP$ determination

The CE method is a versatile new technique for rare event simulation and combinatorial optimization. The method was proven capable of solving a large variety of estimation and

optimization problems, especially NP-hard combinatorial deterministic and stochastic (noisy) problems. We present only the necessary formulas to develop our algorithm. For a comprehensive explanation of the theory and application of CE, refer to the Cross-entropy home page <http://www.cemethod.org> (Lieber, 1998; Rubinstein, 1999; Margolin, 2002; Rubinstein and Kroese, 2004).

In this section, we develop the sample generation method and the main CE algorithm for the CONPIP model.

Consider the following deterministic minimization problem: Let  $X$  be a finite set of states (i.e., all the *NPIP* combinations),  $x \in X$  a particular vector of *NPIP* values, one for each project type, and let the performance function  $S(x)$  be defined as the average total stay time of a project in the system. Suppose we wish to find the minimum of  $S$  over  $X$ , and the corresponding state(s) at which this minimum is attained:

$$S(x^*) = \gamma^* = \underset{x \in X}{\text{minimize}} S(x), \quad (1)$$

where  $\gamma^*$  denotes the minimum. Since we are dealing with stochastic systems, we have to replace  $S(x)$  in (1) with the performance estimator,  $\hat{S}(x)$

$$\hat{S}(x) = S(x) + \zeta(x), \quad (2)$$

where  $\zeta(x)$  is a noise factor.

As stated in Law and Kelton (1991), in complex systems the common approach to estimate  $\mathbb{E}\hat{S}(x)$  (notated later as  $\bar{S}(x)$ ) through simulation experiments. In our case, we shall estimate  $\mathbb{E}\hat{S}(x)$  through the steady state average of projects' stay time after truncation of the warm-up period.

### 2.1. Sample generation

When we have a single project type, a simple method to generate a random state,  $x \in \mathbf{X}_L, \dots, \mathbf{X}_M$  (where  $L$  is the lower bound for *NPIP* value,  $M$  is the upper bound and the jumps between the values are determined according to some desired precision) is to draw  $x$  from a chosen initial discrete distribution vector  $(v_L^0, \dots, v_M^0)$ . When prior knowledge of the real distribution is missing, we use the discrete uniform distribution to generate the initial state.

For the case of  $K$  project types, we denote  $NPIP_i$  as the *NPIP* value for project type  $i$ . Thus, the initial generation of a state vector  $(NPIP_1, \dots, NPIP_K)$  amounts to an independent drawing of  $NPIP_i$  value from a chosen initial discrete distribution vector  $(v_{iL}^0, \dots, v_{iM}^0)$  for all  $i = 1, \dots, K$  and the probabilities to draw *NPIP* values can be arranged in a  $(K) * (M - L + 1)$  matrix,  $V$ .

## 2.2. Main CE algorithm

Following Rubinstein and Kroese (2004), we develop the probability-updating rule for the general case of  $K$  project types. The pdf (probability density function)  $f(\cdot; V)$  is given by:

$$f(\mathbf{X}; V) = \prod_{i=1}^K \sum_{j=L}^M v_{ij} I_{\{\mathbf{X}_i=j\}} \quad (3)$$

where  $I_{\{\mathbf{X}_i=j\}}$  is an indicator function that gets 1 if  $\mathbf{X}_i = j$ , 0 otherwise. Noting that the sum of each row in  $V$  is 1, and using Lagrange multipliers, we get the probability-updating rule:

$$v_{ij} = \frac{\mathbf{E}_v I_{\{\hat{\delta}(\mathbf{X}) \leq \gamma\}} I_{\{X_i=j\}}}{\mathbf{E}_v I_{\{\hat{\delta}(\mathbf{X}) \leq \gamma\}}}, \quad (4)$$

where  $I_{\{\hat{\delta}(\mathbf{X}) \leq \gamma\}}$  are indicator functions on  $\chi$  for various *threshold levels*  $\gamma \in \mathbb{R}$ .

The estimator for the associated probability at the  $t > 0$  iteration is,

$$\hat{v}_{ij}^t = \frac{\sum_{n=1}^N I_{\{\hat{\delta}(\mathbf{X}_n) \leq \gamma_t\}} I_{\{\mathbf{X}_{ni}=j\}}}{\sum_{n=1}^N I_{\{\hat{\delta}(\mathbf{X}_n) \leq \gamma_t\}}}. \quad (5)$$

The sequence of matrices  $V^0, V^1, \dots$  is found to converge to a degenerate matrix  $V^\infty$  containing a single value of 1 in each row (where the rest of the values in each row are 0's). Specifically, we define the stopping rule according to the following two conditions:

$$\varphi_i^t(j) = \varphi_i^{t-1}(j) = \dots = \varphi_i^{t-c}(j) \quad \text{and} \quad v_{ij}^t \geq p', \quad \forall i = 1, \dots, K, \quad (6)$$

where  $c$  is some integer,  $\varphi_i^t(j)$  denotes the index of the maximal element in the  $i$ th row of matrix  $V^t$  and  $p'$  is a probability criterion, whose purpose is to assure convergence to a single combination of *NPIP* values (this may be particularly important if we deal with a multi-extremal performance function in a noisy system where one or more of the values of the local extremums are close to the global extremum value). The sample size,  $N$ , needed to estimate  $(K) * (M - L + 1)$  components of matrices,  $V^t$ , is of the order  $K * (M - L)$  replications (see Rubinstein and Kroese, 2004, Chapter 6 on Noisy optimization networks).

These results lead to the following algorithm.

### Algorithm for CONPIP

1. Generate an initial matrix of probabilities,  $V^0$ , from a discrete uniform distribution  $U[L, \dots, M]$ . Set  $t = 1$  (level counter).
2. Generate a random sample of state vectors  $\mathbf{X}_1, \dots, \mathbf{X}_N$  from  $f(\cdot; V^{t-1})$ . Generate (through simulation experiments) a series of estimators for the performance function

$\{\hat{S}(\mathbf{X}_n) \forall n = 1, \dots, N\}$  and order them in an increasing order:  $\hat{S}_{(1)} \leq \dots \leq \hat{S}_{(N)}$ .  
Let  $\gamma$  be the  $\rho$  sample quantile of performances:  $\hat{\gamma}_t = \hat{S}_{(\lceil \rho N \rceil)}$ .

3. Use the same sample  $X_1, \dots, X_N$  to estimate the optimal  $V^t$  via (5).
4. Stop if convergence is reached according to (6) (let  $T$  denote the final iteration). Otherwise, increase  $t$  by 1 and reiterate from step 2.

We note that instead of updating the parameter  $\hat{v}^{t-1}$  to  $\hat{v}^t$  directly, a smoothing procedure can be used,

$$\hat{v}^t = \alpha \hat{w}^t + (1 - \alpha) \hat{v}^{t-1}, \quad (7)$$

where  $\hat{w}^t$  is the vector derived directly via applying formula (5). This smoothing procedure prevents stopping at a local optimum. Values of  $0.7 \leq \alpha \leq 0.9$  have been found empirically to give the best results (Rubinstein and Kroese, 2004) (it is important to note that the optimal value of the smoothing parameter essentially depends upon a stochastic system's features and values. A possible approach for finding the optimal values of the CE parameters is suggested in the Section 4).

### 3. The experiments

#### 3.1. Description of the experiments

We conducted two types of experiments: In the first experiment, a stochastic, dynamic system processed a single project type. This experiment allowed us to demonstrate that the CE method reached the optimal solution and to gain some insight regarding the behavior of the system under CONPIP management methodology. In the second experiment, the system processed three different project types. This experiment demonstrated the performance of the CE algorithm for a more complicated and noisy system.

#### 3.2. Discussion of the experimental environment

Our experimental tool is a simulation model, written in Visual Basic. Each simulation run started with a long enough warm-up period (i.e., its duration allowed the processing of several thousand projects) that led to a steady state. This transient phase was discarded from the analysis. The system was then simulated for a predefined time interval that was large enough to process several thousand projects. After the CE algorithm converged to a solution (i.e., *NPIP* values), we estimated the performance function value using 100 replications (in Section 3.4.2 we are addressing the effect of reducing the number of replications to estimate the performance function value in a system with three types of projects).

For each one of the experiment types, we generated 10 independent solutions. Then we calculated an *alpha-level* (we used  $\alpha = 0.05$ ) *confidence interval* (CI), given by:

$$\bar{S}(x^*) \pm t_{n-1, \frac{1-\alpha}{2}} \frac{\hat{\sigma}}{\sqrt{n}}, \tag{8}$$

where  $\pm t_{n-1, \frac{1-\alpha}{2}} \frac{\hat{\sigma}}{\sqrt{n}}$  is the *confidence interval variation* (CIV),  $\bar{S}(x^*)$  is the estimator for the optimal performance function value (based on the average of 10 independent solutions) and  $\hat{\sigma}^2$  is the estimator for the variance,

$$\hat{\sigma}^2 = \frac{\sum_{j=1}^n (\hat{S}_j(x) - \bar{S}(x))^2}{n - 1}. \tag{9}$$

The other performance measures that we used to evaluate the CE method performance were:  $\bar{T}$  the mean number of iterations until convergence, the coefficient of variation in percent ( $CV = \frac{\hat{\sigma}}{\bar{S}} * 100$ ), the worst and best absolute relative error, amongst the 10 independent solutions in percentages ( $\varepsilon_*$  and  $\varepsilon^*$ , respectively) and the mean CPU time required for convergence (in seconds).

All the experiments were conducted on a PC platform with an Intel, Pentium 4, 1.8 GHz processor.

### 3.3. Single project type

We applied the CONPIP Algorithm to the stochastic network described in figure 1. The network represents a single project type arriving randomly according to a Poisson process with arrival rate of  $\lambda = 1/3.5$  projects per time unit. Following the approach of Adler et al. (1995), each network node represents a group of (one or more) identical resources performing the same kind of activities. The resources can work in parallel at a

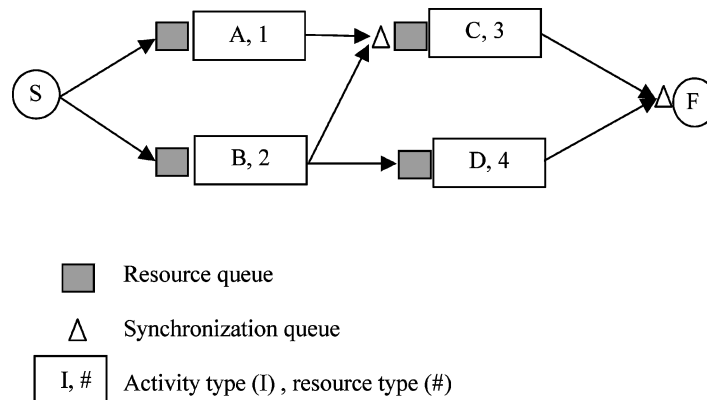


Figure 1. Stochastic processing network representing a multi-project system with a single project type.

Table 1  
Multi-project system characteristics – resource allocation, processing times and time distributions.

	Resource type	Number of resources allocated	Time distribution
Activity A	1	3	$\sim \exp(\mu_A = 1/6)$
Activity B	2	2	$\sim \exp(\mu_B = 1/5)$
Activity C	3	3	$\sim \exp(\mu_C = 1/4)$
Activity D	4	1	$\sim \exp(\mu_D = 1/3)$

processing rate of  $\mu_i$  (processing time taken from exponential distribution  $\sim \exp(\mu_i)$ ) for each resource engaged in activity type  $i$ . The start and finish activities are milestones—they have neither duration nor a resource requirement. Processing time parameters and resource allocations are given in Table 1. Following Anavi-Isakow and Golany (2003), long waiting times for resources are penalized with a two-step penalty function: For waiting times longer than 10 times the average processing time, the penalty is 0.25 times the average processing time and for waiting times longer than 15 times the average processing time, the penalty is 0.5 times the average processing time.

### 3.3.1. The experimental design

The set of parameters chosen is:

We selected  $c = 5$  and  $p' = 0.99$  for our stopping rule (6). We took  $\rho = 0.15$  (in Rubinstein and Kroese (2004) an interval of  $0.01 \leq \rho \leq 0.2$  is suggested to achieve similar results for a buffer allocation problem in a stochastic and dynamic network) and the value of the smoothing parameter,  $\alpha$ , in (7) was set to 0.8 (which is in the recommended interval,  $0.7 \leq \alpha \leq 0.9$ , see Section 2). We set the *NPIP* value interval to  $[1, \dots, 20]$  and fixed the sample size  $N = 100$  (using the formula  $K * (M - L)$  that appeared in Section 2.2 with a factor of 5 to represent an average order of magnitude, i.e.,  $5 * 1 * 20 = 100$ ). We generated 10 independent solutions.

### 3.3.2. Results

All the solutions converged to *NPIP* value of 9, i.e., the number of projects to be processed concurrently in order to minimize the average total stay-time of a project is 9. Figure 2 shows the dynamics of  $v^t$  convergence for a typical solution (in the single project case, vector  $v^t$  represents the degenerated matrix  $V^t$ ). All the solutions converged in 5 to 7 iterations with an average number of 6.1 iterations before convergence. We note that due to the requirement imposed through our selection of  $c = 5$ , the minimal number of iterations we could expect was 5. Hence, convergence was achieved in a very efficient manner in all our experiments. Also, relaxing the convergence test by selecting a smaller value for  $c$  (say,  $c = 3$ ), one can expect that the number of iterations will be reduced.

To verify the quality of the solution, we performed multiple simulations for each *NPIP* value in the range  $[1, 2, \dots, 20]$ . Figure 3 shows that indeed the *NPIP* value of 9 projects gives the minimum total stay-time. Table 2 lists the performance deterioration

$$\begin{aligned}
 v_0 &= (0.050 \ 0.050 \ 0.050 \ 0.050 \ 0.050 \ 0.050 \ 0.050 \ 0.050 \ 0.050 \ 0.050 \ 0.050 \ 0.050 \ 0.050 \ 0.050 \ 0.050 \ 0.050 \ 0.050 \ 0.050 \ 0.050 \ 0.050) \\
 v_1 &= (0.010 \ 0.010 \ 0.010 \ 0.010 \ 0.010 \ 0.010 \ 0.010 \ 0.067 \ 0.296 \ 0.296 \ 0.181 \ 0.010 \ 0.010 \ 0.010 \ 0.010 \ 0.010 \ 0.010 \ 0.010 \ 0.010 \ 0.010) \\
 v_2 &= (0.002 \ 0.002 \ 0.002 \ 0.002 \ 0.002 \ 0.002 \ 0.002 \ 0.002 \ 0.071 \ 0.631 \ 0.231 \ 0.036 \ 0.002 \ 0.002 \ 0.002 \ 0.002 \ 0.002 \ 0.002 \ 0.002 \ 0.002) \\
 &\vdots \\
 v_5 &= (0.000 \ 0.000 \ 0.000 \ 0.000 \ 0.000 \ 0.000 \ 0.000 \ 0.000 \ 0.002 \ 0.997 \ 0.002 \ 0.000 \ 0.000 \ 0.000 \ 0.000 \ 0.000 \ 0.000 \ 0.000 \ 0.000 \ 0.000)
 \end{aligned}$$

Figure 2. A system with a single project type (see figure 1). Dynamics of vector  $v^t$  convergence to  $NPIP$  value equals 9 ( $\lambda = 1/3.5$ , processing times exponentially distributed,  $\mu_A = 1/6$ ,  $\mu_B = 1/5$ ,  $\mu_C = 1/4$ ,  $\mu_D = 1/3$ ; resource allocation for activity types A, B, C, D was taken as 3, 2, 3, 1, respectively).

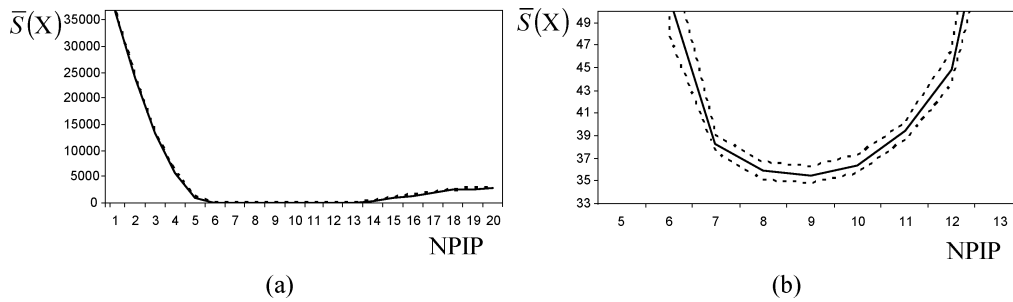


Figure 3. CONPIP model, average project stay-time as a function of the loading parameter value ( $NPIP$ ), (a) is “low resolution” and (b) is “high resolution” with focus to the minimum zone. The dotted lines are the upper and lower 95% confidence interval for the average project stay-time.

(i.e., increase of average project stay-time) as one chooses a non-optimal value of  $NPIP$ . The performance function curve (figure 3) resembles the curve reported in Anavi-Isakow and Golany (2003). We can characterize three different system performance patterns as a function of the  $NPIP$  value setting: (1) The catastrophic deterioration zone: choosing  $NPIP$  value lower than 6 or higher than 13 would lead to a catastrophic deterioration in system performance. (2) The significant deterioration zone: for  $NPIP$  values of 6–7 (LH of the optimum) and 11–13 (RH of the optimum), the deterioration in the performance is in the range of 9% to 85%. (3) The indifferent zone: setting the  $NPIP$  to values 8–10 gives very similar results to the optimum – which means that for most practical applications, it wouldn’t matter if we were to choose  $NPIP$  values in this range rather than the optimal value.

We summarize our numerical experiment as follows:

- (1) We found the optimal  $NPIP$  value using CE.

Table 2  
Average project stay-time for various *NPIP* values.

<i>NPIP</i>	Average project stay-time	% increase in stay-time relative to minimum
1	36,613.2	104,211
2	23,898.5	67,987
3	13,283.3	37,744
4	5,734.7	16,238
5	1,113.3	3,071
6	51.3	46
7	38.3	9
8	35.9	2
9	35.1	0
10	36.4	4
11	39.4	12
12	44.8	28
13	64.8	85
14	265.4	656
15	951.8	2,612
16	1,418.0	3,940
17	1,927.3	5,391
18	2,397.2	6,730
19	2,640.3	7,422
20	2,847.0	8,011

- (2) The average total number of iterations needed in order to find the solution was 6.1. With a sample size of 100, the average total number of replications is 610. If we choose to perform simulations for each *NPIP* value (say, 100 replications for each value), the total number of replications needed to find the optimal *NPIP* value is 2,000 (See Table 3 for summary results).

### 3.4. Three project types

We applied the CONPIP Algorithm to the stochastic network described in figure 4. The network represents three project types being processed by four resource types. Each project type arrives randomly according to a Poisson process with mean inter-arrival rates,  $\lambda_1 = 1/5.5$ ,  $\lambda_2 = 1/9.2$  and  $\lambda_3 = 1/13.8$ , projects per time unit. The processing time parameters and the resource allocations are identical to the case of the single project type. As before, long waiting times for resources are penalized with a two-step penalty function: For waiting times longer than 10 times the average processing time, the penalty is 0.5 times the average processing time, and for waiting times longer than 15 times the average processing time, the penalty is 0.8 times the average processing time.

Table 3

Performance of the CONPIP algorithm for: (a) A system with a single project type arriving according to a Poisson process with  $\lambda = 1/3.5$ ; processing times exponentially distributed with rates  $\mu_A = 1/6, \mu_B = 1/5, \mu_C = 1/4, \mu_D = 1/3$ ; resource allocation for activity types A, B, C, D is 3, 2, 3, 1, respectively. (b) A system with three project types, each one arriving independently according to Poisson process with  $\lambda_1 = 1/5.5, \lambda_2 = 1/9.2$  and  $\lambda_3 = 1/13.8$ ; processing times exponentially distributed with rates,  $\mu_A = 1/6, \mu_B = 1/5, \mu_C = 1/4, \mu_D = 1/3$ ; resource allocation for activity types A, B, C, D is 3, 2, 3, 1, respectively. (c) A system with three project types with the same characteristics as (b) apart from taking the simulation length three times longer than (b).

Case	$\bar{T}$	NPIP	$\bar{S}(x^*)$	CIV	$\hat{\sigma}$	CV	$\varepsilon_*$	$\varepsilon^*$	CPU
(a) Single project type	6.1	9	35.059	$\pm 0.211$	0.295	0.841	1.442	0.009	6,111
(b) Three project types-short simulation	9.4	6,4,3	57.110	$\pm 0.413$	0.577	1.010	2.057	0.006	17,578
(c) Three project types-long simulation	6.8	6,4,3	57.236	$\pm 0.332$	0.464	0.811	1.293	0.276	26,597

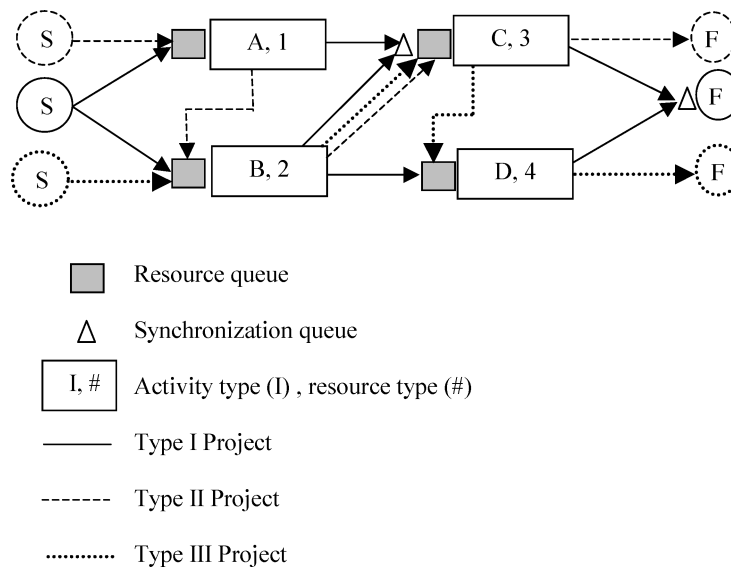


Figure 4. Stochastic processing network representing a multi-project system with three project types.

3.4.1. The experimental design

The experimental design consisted of two stages. In the first, the *calibration stage*, we inspected CE performance under different combinations of parameter values (i.e., different values of  $c, \rho, \alpha$  and sample size). In the second stage, we generated 10 independent solutions under the following CE parameters:  $c = 5$  and  $p' = 0.99$  for our stopping rule (6). We took  $\rho = 0.04$  (which is within the interval of  $0.01 \leq \rho \leq 0.2$  that is suggested in Rubinstein and Kroese (2004) for a buffer allocation problem in a stochastic and dynamic network). The value of the smoothing parameter,  $\alpha$ , in (7) was set to 0.7 (we address

the effect of changing the smoothing parameters' value in the next subsection). We set the *NPIP* value interval for each project type to  $[1, \dots, 20]$  and fixed the sample size  $N = 420$  ( $7 * K * (M - L) = 7 * 3 * 20$ , see Section 2).

3.4.2. Results

All the solutions converged to  $x^* = (6, 4, 3)$ , i.e., to the maximum allowed number of 6 projects of type I, 4 projects of type II and 3 projects of type III, concurrently in process. Figure 5 shows the dynamics of  $V^t$  convergence for a typical solution. Convergence was achieved in 6 to 14 iterations with mean of 9.4 iterations and mean CPU time of 17,578 seconds. Recall that this convergence was achieved using  $\alpha = 0.7$ . Clearly, with smaller  $\alpha$  values, convergence would have taken longer to achieve. To verify this aspect, we ran the experiment again with  $\alpha = 0.6$ . The algorithm converged to the same solution, as expected, within a mean number of 12.6 iterations and a mean CPU time of 23,627 seconds. When the value of  $\alpha$  was further reduced to 0.5, convergence occurred within a mean number of 17.7 iterations and a mean CPU time of 33,568 seconds. It should be noted that over 99% of this CPU time is spent on the simulation part of the procedure while the other steps of the CE algorithm require very little CPU time. For comparison purposes, suppose we were to evaluate all 8,000 alternatives through an exhaustive search technique. Assuming again 100 replications per each alternative and using the value of 4.4 seconds – average simulation time per replication – this would have taken 3,520,000 seconds (more than 40 days) to complete. We note here that one can use, of course, a smaller number of replications to estimate the performance function value at the price of reducing the accuracy. Particularly, for the system with 3 project types, reducing the number of replications from 100 to 50 causes a 240% increase in the standard deviation of the average performance value from 0.413 to 1.404, respectively (CV increases from about 1% to 2.5%). Further reduction to 10 replications causes the standard deviation to increase by more than 1000% with respect to the case of using 100 replications (CV increases now to about 8.2%).

$$\begin{aligned}
 V^0 &= \begin{pmatrix} 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 \\ 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 \\ 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 & 0.050 \end{pmatrix} \\
 &\quad \vdots \\
 V^3 &= \begin{pmatrix} 0.001 & 0.001 & 0.001 & 0.001 & 0.052 & 0.924 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 \\ 0.001 & 0.001 & 0.449 & 0.415 & 0.100 & 0.001 & 0.008 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 \\ 0.001 & 0.008 & 0.450 & 0.001 & 0.001 & 0.001 & 0.285 & 0.001 & 0.001 & 0.001 & 0.092 & 0.001 & 0.014 & 0.073 & 0.001 & 0.001 & 0.021 & 0.001 & 0.021 & 0.021 \end{pmatrix} \\
 &\quad \vdots \\
 V^9 &= \begin{pmatrix} 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 1.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.998 & 0.002 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 1.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \end{pmatrix}
 \end{aligned}$$

Figure 5. A system with three project types (see figure 4). Dynamics of matrix  $V^t$  convergence to  $x^* = (6, 4, 3)$  ( $\lambda_1 = 1/5.5, \lambda_2 = 1/9.2$  and  $\lambda_3 = 1/13.8$ ; processing times exponentially distributed with rates,  $\mu_A = 1/6, \mu_B = 1/5, \mu_C = 1/4, \mu_D = 1/3$ ; resource allocation for activity types A, B, C, D was taken as 3, 2, 3, 1, respectively).

Next, we tripled the simulation length, leaving all other parameters unchanged, to investigate the effect of reducing the noise level over the CE convergence behavior. This time all solutions converged to the same solution  $x^* = (6, 4, 3)$  within 6–7 iterations (mean 6.8) but with higher CPU time (a mean value of 26,597 seconds as compared to 17,578 seconds previously). This may suggest that there is a trade-off between the noise level in the performance function (e.g., shorter simulations result in higher noise level but in lower CPU time per simulation) and the number of replications until convergence.

We conclude the experimental results as follows:

- (1) The total number of possible solutions for this system is 8,000. Since this is a noisy system and because we choose 100 replications to evaluate each solution, we end up with a maximum of 800,000 replications to find the optimal solution. The CE method enabled convergence in 9.4 or 6.8 iterations for the “short” and “long” simulation length, respectively. Each iteration included a sample size of 420, which sums up to convergence in 3,948 or 2856 replications (0.49% or 0.36% of the maximum replications number) for the “short” and “long” simulations, respectively.
- (2) There is a trade-off between the noise level and the convergence rate. Longer simulations may result in a lower number of iterations to convergence than shorter simulations. However, if we make those “long” simulations longer than necessary, the overall time to convergence will be longer, despite the fact that the overall number of replications is lower.
- (3) Table 3 includes summary of results and CE performance.

#### 4. Conclusions

Finite capacity, stochastic, dynamic, multi-project systems are difficult to manage by traditional project management tools. This paper introduces a new self-adjusted approach for managing such systems.

Based on the CONPIP management methodology in which the system is controlled *through keeping a constant number of projects concurrently in the system* a practical procedure for choosing the number of projects in process was developed. The approach finds the CONPIP that minimizes the projects’ average stay-time. Managing the multi-project system using the optimal loading parameter improves the system performance significantly. From a practical point of view, the main advantages of using this approach for estimation of loading parameters lies in the fact that (i) it can handle “noisy” systems effectively, and (ii) that it is self adjusted. We demonstrated via two numerical examples the efficiency of the CE method in finding the solution.

Further research is recommended for investigating the effect of the noise level on convergence speed and on the selection of the CE parameters. Typically, simulations of large, complicated systems are time consuming, so efforts should be directed at finding such parameters that would achieve convergence in the most efficient way, i.e., the

minimum CPU time. Such research can start by investigating different case studies of noisy systems using a two-staged methodology. In the first stage, one will identify the solution (e.g., the *NPIP* value) using the CE method with “conservative” parameters (i.e., low value of smoothing parameter, long simulations, large sample size and large elite sample). The second stage will use the CE method and the information that we found in the first stage to find the values for the different parameters (smoothing parameter, sample size, simulation length etc.) that minimizes a new performance function (for example, CPU time for convergence). The hope is that we can draw some general conclusions about the selection of the different parameters for CE and the noise level.

We believe that the CE method may be applied to the very popular CC multi-project management methodology. As CC methodology aims at developing a sound schedule, using buffer management in order to avoid schedule overruns, it does not provide quantitative models or a standard way to set the load in the multi-project system (i.e. the release rate of projects and the capacity of the buffers). To overcome this shortcoming, we suggest estimating the loading parameters through the CE method. This process may involve two stages: the first is to create the necessary infrastructure to calculate/estimate the performance function value. In our case, this infrastructure may be a simulation program of the multi-project system. The second stage is the application of the CE method. For this case, we may generate the buffer capacity values from a continuous distribution (i.e., Beta distribution starting with  $\alpha = \beta = 1$  which is similar to a uniform distribution or the Normal distribution with initial large standard deviation). The performance function may be determined as minimum lateness of projects with respect to a predetermined due-date or minimum average duration of projects. The hope is that as convergence is reached, the loading parameters would be set to their optimal values.

### Acknowledgments

We would like to thank Professor Reuven Rubinstein from the Technion – Israel Institute of Technology for his valuable suggestions on an earlier draft of the paper.

### References

- Abdel-Hamid, T. and S.E. Madnick. (1991). *Software Project Dynamics: An Integrated Approach*. Englewood Cliffs: Prentice Hall.
- Adler, P.S., A. Mandelbaum, V. Nguyen, and E. Schwerer. (1995). “From Project to Process Management: An Empirically-Based Framework for Analyzing Product Development Time.” *Management Science* 41(3), 458–484.
- Adler, P.S., A. Mandelbaum, V. Nguyen, and E. Schwerer. (1996). “Getting the Most Out of Your Product Development Process.” *Harvard Business Review* March-April, 134–152.
- Anavi-Isakow, S. and B. Golany. (2003). “Managing Multi-Project Environments Through Constant Work in Process.” *International Journal of Project Management* 21(1), 9–18.
- Archibald, R.D. and R.L. Vitoria. (1967). *Network-Based Management Systems (PERT/CPM)*. New York: John Wiley & Sons.

- Banker, R.D., S.M. Datar, and S. Kekre. (1988). "Relevant Costs, Congestion and Stochasticity in Production Environments." *Journal of Accounting and Economics* 10, 171–197.
- Cross-entropy home page, <http://www.cemethod.org>.
- Davis, E.W. (1974). "CPM Use in Top 400 Construction Firms." *Journal of the Construction Division* 100(1), 39–49.
- Goldratt, E.M. (1997). *Critical Chain*. USA: The North River Press.
- Herroelen, W. and R. Leus. (2001). "On the Merits and Pitfalls of Critical Chain Scheduling." *Journal of Operations Management* 19(5), 559–577.
- Hopp, W.J. and M.L. Spearman. (1996). *Factory Physics - Foundations of Manufacturing Management*. USA: Irwin.
- Karmarkar, U.S. and S. Kekre. (1992). "Multi-Item Batching Heuristics for Minimization of Queuing Delays." *European Journal of Operational Research* 58, 99–111.
- Kekre, S. (1987). "Performance of a Manufacturing Cell with Increased Product Mix." *IIE Transactions* 19(3), 329–339.
- Kuik, R. and P.F.J. Tielemans. (2003). "Expected Time in System Analysis of a Single-Machine Multi-Item Processing Center." *European Journal of Operational Research* (in press).
- Kurtulus, I. and E.W. Davis. (1982). "Multi Project Scheduling: Categorization of Heuristic Rules Performance." *Management Science* 28(2), 161–172.
- Law, A.M. and W.D. Kelton. (1991). *Simulation Modeling and Analysis*. USA: McGraw-Hill.
- Leach, P.L. (1999). "Critical Chain Project Management Improves Project Performance." *Project Management Journal* 30(2), 39–51.
- Lieber, D. (1998). "Rare-Events Estimation via Cross-Entropy and Importance Sampling." Ph.D. Thesis, William Davidson Faculty of Industrial Engineering and Management, Technion, Haifa, Israel.
- Margolin, L. (2002). "Application of the Cross-Entropy Method to Scheduling Problems." M.Sc. Thesis, William Davidson Faculty of Industrial Engineering and Management, Technion, Haifa, Israel.
- Matsuura, H., H. Tsubone, and M. Kanazashi. (1996). "Setting Planned Lead Times for Multi-Operation Jobs." *European Journal of Operational Research* 88(2), 287–303.
- Rubinstein, R.Y. (1999). "The Cross-Entropy Method for Combinatorial and Continuous Optimization." *Methodology and Computing in Applied Probability* 1, 127–190.
- Rubinstein, R.Y. and D.P. Kroese. (2004). *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Neural Computation*. Springer-Verlag, New York.
- Sprezza, M.G. and C. Vercellis. (1993). "Hierarchical Models for Multi Project Planning and Scheduling." *European Journal of Operational Research* 64(2), 312–325.
- Tielemans, P.F.J. and R. Kuik. (1996). "An Exploration of Models that Minimize Leadtime Through Batching of Arrived Orders." *European Journal of Operational Research* 95(2), 374–389.
- Yang, J. and R.H. Deane. (1993). "Setup Time Reduction and Competitive Advantage in a Closed Manufacturing Cell." *European Journal of Operational Research* 69, 413–423.