Application of the Cross Entropy Method for Buffer Allocation Problem in Simulation Based Environment

G. Allon, T. Raviv and R. Y. Rubinstein

Faculty of Industrial Engineering and Management, Technion, Haifa, Israel

Abstract

This paper deals with application of the *cross-entropy* (CE) method for solving the *buffer allocation problem* (BAP).

The CE method for combinatorial optimization presents an adaptive algorithm equipped with an auxiliary random mechanism for transforming the original deterministic problem into a stochastic one. For the BAP the random mechanism presents an $(m-1)$ x $(\mathrm{n}+1)$ probability matrix $\boldsymbol{P} = (P_{ij})$, such that $\sum_{j=0}^{n} P_{ij} = 1, \forall i = 1, \ldots, m-1$, where $m-1$ and n denote the number of niches (the number of machines is the number of niches $+1$) and the amount of buffer spaces, respectively. The P_{ij}-th element of $\boldsymbol{P}$ represents the probability of niche i receiving j buffer spaces (note that allocation of 0 buffers in some niche is feasible). Once the matrix $\boldsymbol{P}$ is defined, each iteration of the CE algorithm comprises of the following two phases:

1. Generation of random buffer allocations using the auxiliary probability matrix $\boldsymbol{P} = (P_{ij})$ and simultaneous calculation of an objective function, such as the expected steady-state throughput of the system or its sample equivalent obtained via simulation.
2. Updating of the parameters of the probability matrix (P_{ij}) on the basis of the data collected in the first phase. This is done via the simulated cross-entropy method.

We show numerically that after some finite number of iterations the initial auxiliary probability matrix $\boldsymbol{P}$ becomes a *degenerated* one, denoted as $\boldsymbol{P}_d^*$, such that only a *single* element of each column in it equals unity, while the remaining elements are zeros. The unity elements of $\boldsymbol{P}_d^*$ will define (estimate) the optimal buffer allocation.

We shall also show numerically that our algorithm converges in a finite number of iterations with very high probability, to a very small subset of the closest to the optimal values, one of which is the optimal solution itself.

Keywords. Buffer Allocation, Cross-entropy, Importance Sampling, Production Lines, Rare Events, Simulation

1 Introduction

This work deals with application of the *cross-entropy* (CE) method introduced in [8]- [11] for solving the *buffer allocation problem* (BAP).

In asynchronous production lines, when a machine finishes processing a certain job and the next buffer has available space, the processed job is passed on and the job located in the preceding buffer is taken up. If either the next buffer is full or the preceding buffer is empty, the machine remains idle until the situation changes, i.e. when either the preceding or the next machine finishes processing its jobs. These situations are called *blocking* and *starving*, respectively.

In a BAP model, one typically assumes that

1. The first machine is never starved and the last one is never blocked, but both the raw material and the finished-product storage are unlimited.

2. Each machine has its own processing time assumed to be a random variable with some given distribution function. The processing times of the machines are mutually independent.

For a given production line with m machines and b buffer spaces, we have $\binom{n+m-2}{m-1}$ feasible allocations, which clearly column exponentially in both the number of machines and buffer spaces.

Two common performance measures for production lines are the expected *throughput* of the line and the expected quantity of *work-in-process* (WIP). As we shall see below, the *cross-entropy* (CE) method is capable of optimizing this performance and their noisy equivalents obtained via simulation.

We distinguish further between analytic and simulation-based methods for performance evaluation. Note that exact (analytical) methods for evaluation of the expected steady-state performance measures in BAP's are available only for exponential and phase type distribution [1], [3], while simulation based models are, in fact, distribution independent. By simulation we mean standard discrete-event simulation [6], [7] that estimates the performance, like throughput or WIP, for a fixed buffer allocation (configuration) among the niches. We shall solve the BAP in the steady-state since production lines are intended for large amounts of identical products. Although CE can handle successfully both deterministic and noisy NP-hard *combinatorial optimization problems* (COP) [11], our main goal here is the noisy type, namely simulation-based optimization.

For a production line with m workstations, let us denote the the intermediate locations for the buffers by $B_1, \ldots, B_{m-1}$. Let also $x_i = x(B_i)$ denote the number of the buffer allocated in the i^{th} niche. Our goal is to maximize the mean steady-state objective function, such as the mean steady-state throughput, denoted $\mathcal{M}(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, \ldots, x_{m-1})$ in a production line given a fixed amount of buffer spaces n, that is

$$\begin{aligned} &\max \mathcal{M}(x_1, \ldots, x_{m-1}) \\ &\text{s.t.} \\ &\textstyle\sum_{i=1}^{m-1} x_i = n \end{aligned} \tag{1.1}$$

In the nomenclature of Gershwin and Shor [1] the program (1.1) is called the *dual* problem in distinction from the *primal*, which can be written as

$$\begin{aligned} &\min \textstyle\sum_{i=1}^{m-1} x_i \\ &\text{s.t.} \\ &\mathcal{M}(x_1, \ldots, x_{m-1}) \geq C \end{aligned} \tag{1.2}$$

The program (1.2) reads: Given a required steady-state throughput C, minimize the total amount of buffer spaces to be allocated.

We shall apply here the CE method for the dual problem, which is more natural and more frequently used, leaving the primal as a topic for further investigation.

In addition, we also allow the production lines to be unreliable. In particular, as Vouros and Papadopolous [13] did, we assume that each machine i may have exponentially distributed breakdowns with rate β_i, which occur only when the machines are active, and exponentially distributed up times with rate r_i. The CE method handles this just as a special case.

The rest of the paper is organized as follows. In Section 2 we presents the CE algorithm for the BAP. Section 3 presents some extensions of the CE algorithm. Supportive numerical results are presented in Section 4, and concluding remarks and directions for further research are given in Section 5.

2 The Cross Entropy Algorithm

The CE method for combinatorial optimization [8]- [11] presents an adaptive algorithm equipped with a random mechanism for transforming the original deterministic problem into a stochastic one, called the *associated stochastic problem* (ASP). In the ASP the random mechanism presents an $(m-1)$ x $(n+1)$ matrix $\boldsymbol{P} = (P_{ij})$, $i = 1, \ldots, m-1$; $j = 0, 1, \ldots, n$. Here $m-1$ denotes the number of niches (the space between two machines), and n the total buffer spaces to be allocated to the workstations in some optimal way (say so as to maximize the *expected steady-state throughput* of the system) respectively. The P_{ij}-th element of $\boldsymbol{P}$ represents the probability of niche i receiving j buffer spaces. Note also that $\boldsymbol{P} = (P_{ij})$ is defined as the matrix in an $m-1$-dimensional distribution $\boldsymbol{M}(\boldsymbol{P}) = (M_1(\boldsymbol{P}_1), \ldots, M_{m-1}(\boldsymbol{P}_{m-1}))$ with independent marginals

$M_i(y, \boldsymbol{P}_i) = M(y, P_{i0}, P_{i1}, \ldots, P_{in})$ and such that each $M_i(y),\ i = 1, \ldots m-1;\ y = 0, 1, \ldots, n$ represents an $n+1$-point discrete distribution with the mass at points $y = 0, 1, \ldots, n$. Clearly, that

$$\sum_{j=0}^{n} P_{ij} = 1, \forall i = 1, \ldots, m-1 \tag{2.3}$$

As soon as the ASP is defined, each iteration of the CE method contains the following two phases:

1. Generation of random buffer allocations, using an auxiliary probability matrix $\boldsymbol{P} = (P_{ij})$, and simultaneous calculation of the objective function, such as the expected steady-state throughput or its sample equivalent obtained from simulation, see Algorithm 2.2.

2. Updating the parameters of the probability matrix (P_{ij}) at the end of each iteration. This is done using the cross-entropy Algorithm 2.1 below.

We shall consider how to construct the initial matrix $\boldsymbol{P}$ and the stopping rules for the CE Algorithm 2.1. We shall also show numerically that after some finite number of iterations the initial auxiliary probability matrix $\boldsymbol{P}$ converges to a *degenerated* one, denoted $\boldsymbol{P}_d^*$, such that only a *single* element of each column i of $\boldsymbol{P}_d^* = (P_{d,ij}^*)$ equals unity, while the remaining elements are zeros. The unity elements of $\boldsymbol{P}_d^*$ will define the optimal buffer allocation. We shall finally show numerically that Algorithm 2.1 converges in a finite number of iterations, with very high probability, to a very small subset of the closest-to-the-optimal values, one of which is the optimal solution itself. Supportive numerical results will be given as well. Our numerical studies below suggest that the presented CE method for BAP has polynomial complexity with respect to both parameters m and n.

Algorithm 2.1 Main Algorithm :

1. *Choose an initial $m-1$-dimensional pdf $\boldsymbol{M}(\boldsymbol{P}_0)$ (with the reference $(m-1 \text{ x } n+1)$ matrix $\widehat{\boldsymbol{P}}_0^* \equiv \boldsymbol{P}_0$) and, say with elements uniformly distributed at each column, that is take*

$$P_{0,ij} = \frac{1}{n+1},\ j = 0, 1, \ldots, n,\ \forall i = 1, \ldots, m-1.$$

Using Algorithm 2.2, generate N buffer allocations $\mathcal{Z}_s = \{(j_1, j_2, \ldots, j_{m-1})\};\ s = 1, 2 \ldots, N$, calculate their corresponding throughput $\mathcal{M}(\mathcal{Z}_s)$ and deliver the optimal solution of the program

$$\max\ \gamma_t\ \text{ s.t. }\ \left\{\frac{1}{N}\sum_{s=1}^{N} I_{\{\mathcal{M}(\mathcal{Z}_s) \geq \gamma_t\}}\right\} \geq \rho, \tag{2.4}$$

For $t = 0$. Denote the initial solution by $\bar{\gamma}_0^*$. Set $t = 1$.

2. *Use the* **same** *N random buffer allocations and their corresponding throughputs to create $\widehat{\boldsymbol{P}}^*_t$. as follows.*

$$\widehat{P}^*_{t,kl} = \frac{\sum_{1\le j\le N:\mathcal{Z}_{s,j_k}=l} I_{\{\mathcal{M}(\mathcal{Z}_s)\ge\bar{\gamma}^*_{t-1}\}}}{\sum_{1\le s\le N} I_{\{\mathcal{M}(\mathcal{Z}_s)\ge\bar{\gamma}^*_{t-1}\}}} \tag{2.5}$$

   Note that the denominator of 2.5 is by definition $\lfloor N\rho\rfloor$.

3. *Generate N* **new** *random buffer allocations $\{(j_1, j_2, \ldots, j_{m-1})\}$ from $\widehat{\boldsymbol{P}}^*_t$ using algorithm 2.2 and deliver the optimal solution of the program (2.4). Denote the solution by $\bar{\gamma}^*_t$.*

4. *If $\max_j \boldsymbol{P}^*_{t,ij} > 1-\rho$ for any i then stop and return $x_i = argmax_j \boldsymbol{P}^*_{t,ij} - 1$ for any i, otherwise, set $t = t+1$ and go to step 2. This stopping rule is based on the convergence of $\widehat{\boldsymbol{P}}^*_t$ to $\boldsymbol{P}^*_d$, which is called the optimal degenerated matrix (ODM).*

We present bellow an algorithm for random buffer allocation to be used as a subroutine of the main algorithm2.1. Note that other random mechanisms for random buffer allocation may be used as well.

Algorithm 2.2 Random Generation of Buffer Allocation

1. *Generate a random variable (rv) from the $(m-1)$ point discrete uniform pdf $\mathcal{U}^{(1)}(x)$, $x = 1, \ldots, m-1$. Denote the outcome (niche number) by i_1. Generate a rv from the original $n+1$ point discrete pdf*

$$M_{i_1}(y, P_{i_10}, P_{i_11}, \ldots, P_{i_1n}) \equiv M^{(1)}_{i_1}(y, P_{i_10}, P_{i_11}, \ldots, P_{i_1n}),$$

   $i_1 = 1, \ldots m-1$; $y = 0, 1, \ldots, n$. Denote the outcome (amount of the buffer space assigned to niche i_1) by j_1. Assign an amount j_1, $(j_1 = 0, 1, \ldots, n)$ of buffer space to niche i_1).

2. *If $j_1 = n$, stop and assign no buffer space to the remaining niches i_s, $s = 2, \ldots, m-1$, otherwise modify step 1 as follows. Generate a random variable from the $m-2$ point discrete uniform pdf $\mathcal{U}^{(2)}(x)$, $x = 1, \ldots, i_1-1, i_1+1, \ldots, m-1$. Denote the outcome by i_2. Generate a rv from the (truncated) discrete $n-j_1+1$ point pdf $M^{(2)}_{i_2}(y, P_{i_20}, P_{i_21}, \ldots, P_{i_2(n-j_1)})$, $y = 0, 1, \ldots, n-j_1$ obtained from the original $n+1$- point pdf $M_{i-1}(y, P_{i_20}, P_{i_21}, \ldots, P_{i_2n})$, $y = 0, 1, \ldots, n$ by eliminating (truncating) j_1 values of $y = 0, 1, \ldots, n$ from the right and normalizing the components $P_{i_20}, P_{i_21}, \ldots, P_{i_2(n-j_1)}$ as $\sum_{k=0}^{n-j_1} P_{i_2k} = 1$. Denote the outcome by j_2. Assign an amount of j_2, $(j_2 = 0, 1, \ldots, n-j_1)$ buffer space to niche i_2.*

3. *Proceed with steps 2 recursively until either $\sum_{r=1}^{m-1} j_r = n$ or $\sum_{r=1}^{m-2} j_r = q < n$. In the first case assign no buffer space to the remaining niches and stop, while in the second case assign an amount of buffer space $n-q$ to the remaining niche i_{m-1} and stop.*

Note that the sequence $\{i_1, \ldots, i_{m-1}\}$ of niches generated from the sequence of discrete uniform distributions $\{\mathcal{U}^{(s)}(x),\ s = 1, \ldots m-1\}$ ensures that each niche is selected *at most once.* (If a niche is not selected, then no buffer is allocated to it). Note also that the random walk (buffer allocation sequence among the niches) $\{j_1, \ldots, j_{m-1}\}$ generated from the sequence of discrete distributions $\{M^{(1)}_{i_1}, \ldots, M^{(m-1)}_{i_{m-1}}\}$ ensures that the total available amount of buffer space n is distributed uniquely among the $m-1$ niches.

3 Extensions

In this section we shall consider several extensions and modifications of Algorithm 2.1 with a view to speeding up its convergence.

3.1 Alternative sample functions

A natural modification of Algorithm 2.1 would be to use in the second step the top $\lfloor \rho N \rfloor$ **values** of $\mathcal{M}_1, \ldots, \mathcal{M}_N$, (or $\mathcal{M}_1^\beta, \ldots, \mathcal{M}_N^\beta$) instead of the indicators$I_{\{\mathcal{M}_i > \gamma\}}$, $i = 1, \ldots, N$. So 2.5 is modified to:

$$\widehat{P}^*_{t,kl} = \frac{\sum_{1 \le s \le N: \mathcal{Z}_{s,j_k} = l} \mathcal{M}(\mathcal{Z}_s)^\beta I_{\{\mathcal{M}(\mathcal{Z}_s) \ge \bar{\gamma}^*_{t-1}\}}}{\sum_{1 \le s \le N} \mathcal{M}(\mathcal{Z}_s)^\beta I_{\{\mathcal{M}(\mathcal{Z}_s) \ge \bar{\gamma}^*_{t-1}\}}} \tag{3.6}$$

We found that this modification increases the speed of Algorithm 2.1 by a factor of up to 2-3 times, provided $1 \le \beta \le 2$. The reason is that, using indicators we assign the same (*unity*) weight to each P_{ti} associated with each top $\lfloor \rho N \rfloor$ value of $\mathcal{M}_1, \ldots, \mathcal{M}_N$, while in the modified version we assign a weight proportional to the corresponding value of $\mathcal{M}_i^\beta$ (as before, each updating of P_{ti} is associated with the top $\lfloor \rho N \rfloor$ values of $\mathcal{M}_1, \ldots, \mathcal{M}_N$).

In fact any monotone nondecreasing (nonincreasing) function $\varphi(\mathcal{M})$ for a maximization (minimization) problem can be used instead of the indicator function.

3.2 Smoothed probability matrices

As an alternative to $\widehat{\boldsymbol{P}}^*_t$ (obtained from (2.5)) we can use its following smoothed version

$$\widetilde{P}^*_{t,ij} = \alpha \widehat{P}^*_{t,ij} + (1-\alpha)\widehat{P}^*_{t-1,ij},\ \forall i = 1, \ldots n, \tag{3.7}$$

where $0 < \alpha \le 1$. Clearly, for $\alpha = 1$ we have that $\widetilde{P}^*_{t,ij} = \widehat{P}^*_{t,ij}$.

The purpose of using $\widetilde{P}^*_{t,i}$ instead of $\widehat{P}^*_{t,i}$ is to smooth out the values of the latter and thus reduce probability of some values of $\widehat{P}^*_{t,i}$ being zeros. It is clear that with $\alpha < 1$ the probability of an element of $\widetilde{\boldsymbol{P}}^*_t$ being exactly zero is 0.

We found empirically that with $\alpha = 0.7$ Algorithm 2.1:

1. The probability of convergence to the optimal solution is higher.

2. The algorithm runs two time and more faster than with $\alpha = 1$.

Also, since CE can be viewed as an ant-colony method we can call α and $\widehat{P}^*_{t,i}$ the *evaporation* parameter and *pheromone* matrix, respectively.

4 Numerical Results

In this section we present the performance of Algorithm 2.1. We implemented the algorithm in Matlab 5.2 and ran it on Intel Pentium III 500MHz.

We applied Algorithm 2.1 to 140 problems presented by Vouros and Papadopolous [13] as well to some other BAP. In all 140 problems of [13] the processing time of each operation is assumed to be distributed either exponentially or Erlang with given service rates $\mu_i, i = 1, \ldots, m-1$. The failure rate and repair time are assumed to be distributed exponentially as well with rates β_i and $r_i, \;\; i = 1, \ldots, m-1$, respectively. Vouros and Papadopolous [13] present the exact solution of each of the problems, calculated by evaluating the Markov Chain represented the systems as described in [3]. Each of the 140 problems of [13] was ran 10 times. Some of the results of these 1400 experiments are presented in Tables 4.1 and 4.2, where we used the following notations:

$$\epsilon = 100 \cdot \frac{\gamma^*_b - \gamma^*_T}{\gamma^*_b}, \tag{4.8}$$

where γ^*_T is the estimate of γ^* obtained after stopping, γ^*_b denotes the best known solution, $\bar{\epsilon}$ denotes the average relative error based on 10 replications, that is

$\bar{\gamma}^*_T$ denotes the average percentage of the $\gamma^*_{T_i}, \; i = 1, \ldots, 10$ based on 10 replications, ϵ_* and ϵ^* denote the worse and the best percentage of relative error among the 10 generated solutions, $\bar{T}$ and CPU denote the average total number of iterations needed before stopping and the average CPU time in seconds, respectively.

We generated at each iteration of Algorithm 2.1 $N = 5\sqrt{nm}$ random allocations according to the Algorithm 2.2. We used everywhere $\rho = 0.1$ and $\alpha = 0.7$. Note that similar results were obtained with $0.05 \leq \rho \leq 0.2$ and $0.5 \leq \alpha \leq 1.0$. We found numerically that algorithm 2.1 stops approximately after $O(mn)$ iterations.

Table 4.1 Performance of Algorithm 2.1 for 5-stations with exponential operation rates $\mu_1 = 1$, $\mu_2 = 1.1$, $\mu_3 = 1.2$, $\mu_4 = 1.3$, $\mu_5 = 1.5$, breakdowns rates $\beta_1 = \beta_2 = \beta_3 = \beta_4 = \beta_5 = 0.05$ and up times rates $r_1 = r_2 = r_3 = r_4 = r_5 = 0.5$.

n	$\bar{T}$	$\bar{\gamma}_T^*$	γ_b^*	$\bar{\epsilon}$	ϵ_*	ϵ^*	CPU
1	2.6	.5213	.5213	0	0	0	12
2	4.6	.5479	.5514	0.60	1.10	0	32
3	3.6	.5824	.5824	0	0	0	39
4	6.4	.6015	.6027	0.20	0.85	0	67
5	9.0	.6202	.6213	0.18	0.32	0	103
6	5.7	.6420	.6422	0.03	0.31	0	89
7	7.7	.6572	.6585	0.20	0.87	0	116
8	7.2	.6731	.6744	0.20	1.20	0	132
9	9.1	.6885	.6894	0.13	0.67	0	166
10	10.7	.7004	.7005	0.02	0.03	0	197

Table 4.2 Performance of Algorithm 2.1 for 5-stations with Erlang 2 operation rates $\mu_1 = 1$, $\mu_2 = 1.1$, $\mu_3 = 1.2$, $\mu_4 = 1.3$, $\mu_5 = 1.5$, breakdowns rates $\beta_1 = \beta_2 = \beta_3 = \beta_4 = \beta_5 = 0.05$ and up times rates $r_1 = r_2 = r_3 = r_4 = r_5 = 0.5$.

n	$\bar{T}$	$\bar{\gamma}_T^*$	γ_b^*	$\bar{\epsilon}$	ϵ_*	ϵ^*	CPU
1	2.8	.5968	.5968	0	0	0	23
2	3.5	.6331	.6338	0.11	1.14	0	39
3	3.9	.5824	.5824	0	0	0	55
4	5.8	.6802	.6808	0.09	0.73	0	86
5	8.3	.6985	.6996	0.16	0.28	0	159
6	6.9	.7180	.7195	1.14	0.2	0	187
7	12.5	.7335	.7341	0.18	0.07	0	202
8	9.8	.7496	.7501	0.43	0.07	0	181
9	9.7	.7620	.7627	0.68	0.09	0	177
10	13.6	.7714	7740	1.24	.33	0	261

It follows from the results of Tables 4.1 and 4.2 that the allocations found by the CE method are very close to the best known ones. Note that for 4.1 the best known solution and the optimal solutions are the same, (due to Vouros and Papadopolous [13]).

We obtained similar accuracies for different processing time distributions (i.e., Exponential, Normal, Erlang, Uniform), provided $0.05 \leq \rho \leq 0.2$ and $0.5 \leq \alpha \leq 1.0$. That is we found Algorithm 2.1 is quite robust.

Tables 4.3 presents performance of our algorithm for exponentially distributed processing times and different values of n, $m = 6$.

Table 4.3 Performance of Algorithm 2.1 for 6-stations with Exponential operation rates $\mu_1 = 8$, $\mu_2 = 11$, $\mu_3 = 14$, $\mu_4 = 14$, $\mu_5 = 11$,$\mu_6 = 8$

n	$\bar{T}$	$\bar{\gamma}^*_T$	γ^*_b	$\bar{\epsilon}$	ϵ_*	CPU
2	4.2	5.4935	5.5027	0.17	0.84	32.80
4	5.4	5.9245	5.9334	0.15	0.76	65.80
6	12	6.2443	6.2555	0.18	0.50	156.00
8	13.4	6.5197	6.5253	0.09	0.22	198.80
10	25.6	6.7510	6.7589	0.12	0.57	386.40
12	49	6.9316	6.9360	0.06	0.11	766.40
14	28.2	7.0684	7.0934	0.35	0.79	603.60
16	59.2	7.1783	7.1846	0.09	0.26	1128.60
18	92.6	7.4149	7.4291	0.19	0.36	2048.40

We could not compare the results of table 4.3 since to the best of our knowledge no such case studies are available yet for such relatively large systems. We believe, however, that our results are accurate and reliable and can serve as case studies to compare different heuristics. Note also that γ^*_b in table 4.3 corresponds to our best solution obtained (on the basis of 10 different runs) for each fixed n.

A documented Matlab source cod file is available at
www: http://tx.technion.ac.il/~talraviv/BAP.

5 Concluding Remarks and Guidelines for Further Research

This paper presents an adaptive cross-entropy algorithm for solving the BAP. The CE method can handle successfully both deterministic and stochastic NP-hard *combinatorial optimization problems* (COP) [11]. our main goal here is the noisy type, namely simulation-based optimization.

Our numerical studies suggest that in approximately 99% the relative error ϵ does not exceed 1%. In addition we found numerically that Algorithm 2.1 has polynomial complexity in the size of the network. Some further topics for investigation are listed below.

1. Establish confidence intervals (regions) for the optimal solution.
2. Apply parallel optimization techniques to the proposed methodology.
3. Present more case studies.

References

[1] Gershwin, S.B. and J.E. Schor, (2000), Efficient algorithms for buffer space allocation, *Annals of Operations Research* **93**, 117–144.

[2] Gutjahr, W.J. (2000). A Graph-based Ant System an d Its Convergence, *Future Generations Computing* **16**, 873–888.

[3] Heavey C., Papadopoulos H.T. and J. Browne, *The throughput rate of multistation unreliable production lines*, European Journal of Operation Research, 1993, 68, 69-89.

[4] Lieber, D., Rubinstein, R.Y. and D. Elmakis, "Quick estimation of rare-events in stochastic networks", *IEEE Transactions on Reliability* **46** (2), 254–265, 1997.

[5] Norkin, W.I., G.C. Pflug, and A. Ruszczyński, "A branch-and-bound method for stochastic global optimization". Working paper, International Institute for Applied System Analysis, WP-96-065, Laxenburg, Austria, 1996.

[6] H.T. Papadopoulos, G.A. Vouros, *A model management System (MMS) for the design and operation of production lines* International Journal of production Research, 1997, 35(8), 2213–2236

[7] Rubinstein, R.Y. and B. Melamed, *Efficient Simulation and Modeling*, John Wiley & Sons, 1998.

[8] Rubinstein, R.Y., "The simulated entropy method for combinatorial and continuous optimization", *Methodology and Computing in Applied Probability* **2**, 127–190, 1999.

[9] Rubinstein, R.Y., "Combinatorial optimization via cross- entropy ", S. Gass and C. Harris, Editors, Kluwer, *Encyclopedia of Management Sciences*, 2000

[10] R.Y. Rubinstein (2000b), "The cross-entropy method and rare-events for maximal cut and bipartition problems". Manuscript, Technion, Haifa, Israel.

[11] R.Y. Rubinstein, "Combinatorial Optimization, Cross-Entropy, Ants and Rare Events", S. Uryasev and P. M. Pardalos Editors, Kluwer, *Stochastic Optimization: Algorithms and Applications*, 2001.

[12] Spinelis, D.D, Papadopoulos H.T, Production Line Buffer Allocation: Genetic Algorithms Versus Simulated Annealing, ANNALS OF OR, 2000

[13] G.A. Vouros H.T. Papadopoulos, *Buffer Allocation in Unreliable Production Lines Using a Knowlage Based System*, Computer & Operation Research, 1998, Vol. 25:12, pp. 1055-1067