European Journal of Operational Research
Manuscript Draft

Manuscript Number:

Title: A Cross-Entropy Based Algorithm for Combinatorial Optimization Problems

Article Type: Regular Paper

Section/Category: Metaheuristics

Keywords: Combinatorial Optimization; Metaheuristics; Reliability

Manuscript Region of Origin:

Abstract: Cross Entropy has been recently applied to combinatorial optimization problems with promising results. This paper proposes a new cross-entropy based algorithm for monotonic integer programming problems with knapsack-type constraints. We show how the metaheuristic method can be adapted to tackle constrained problems, provided that some basic mathematical properties are enforced. Furthermore, some insight on the convergence of the method is provided. Finally, the algorithm is tested on a number of problems drawn from the reliability realm. The proposed algorithm has been extensively tested both on problems from the literature and on randomly generated large scale instances. Computational results prove the effectiveness as well as the robustness of the algorithm on different classes of problems.

A Cross-Entropy Based Algorithm for Combinatorial Optimization Problems

M. Caserta [a,*], M. Cabo Nodar [a]

[a] *Instituto Tecnologico de Monterrey, Calle del Puente, 222, Col. Ejidos de Huipulco, Del. Tlalpan, México DF, 14380, México.*

Abstract

Cross Entropy has been recently applied to combinatorial optimization problems with promising results. This paper proposes a new cross-entropy based algorithm for monotonic integer programming problems with knapsack-type constraints. We show how the metaheuristic method can be adapted to tackle constrained problems, provided that some basic mathematical properties are enforced. Furthermore, some insight on the convergence of the method is provided. Finally, the algorithm is tested on a number of problems drawn from the reliability realm. The proposed algorithm has been extensively tested both on problems from the literature and on randomly generated large scale instances. Computational results prove the effectiveness as well as the robustness of the algorithm on different classes of problems.

Key words: Combinatorial Optimization; Metaheuristics; Reliability

1 Introduction

Many real world problems, when modeled, fall into the category of integer and combinatorial optimization problems. However, combinatorial optimization problems are known to be $\mathcal{NP}$-Hard [1] which implies, if $\mathcal{P} \neq \mathcal{NP}$, that there is no polynomial time algorithm for these kind of problems. For this reason, researchers have devoted much attention to the development of approximate, or heuristic-based, approaches for this family of problems.

The most recent trend in combinatorial optimization is focused on the development of metaheuristic algorithms to tackle large scale instances ([2–6]).

* Corresponding author. Tel.: +52 55 5483 2189; fax: +52 55 5483 2163.
Email addresses: `marco.caserta@itesm.mx` (M. Caserta), `marta.cabo@itesm.mx` (M. Cabo Nodar).

Metaheuristics are intelligent strategies able to extract knowledge during the search process. Throughout the search, a metaheuristic algorithm constantly evaluates the solutions obtained with the aim of extracting useful information to improve the future search. An interesting classification of metaheuristic schemes along different dimensions is provided in Blum and Roli [6].

One innovative metaheuristic is known as "cross entropy" method (CE), proposed by Rubinstein ([7–9]). Cross-entropy was proposed as an adaptive algorithm used to estimate *rare event* probabilities within complex networks. In recent years, however, cross-entropy has gained much attention for its ability to effectively tackle combinatorial optimization problems ([10,11]). The Cross-Entropy method has been successfully applied to complex combinatorial optimization problems, such as traveling salesman problems [12], buffer allocation problems [13], vehicle routing problems [14], maximal cut problems [15], and sequence alignment problems in genetics [16].

This paper proposes a new cross-entropy based algorithm for monotonic integer programming problems, with a special emphasis on complex systems reliability optimization problems. We introduce a scheme, called "projection scheme", which makes the algorithm especially suited to solve integer programming problems with knapsack-type constraints. We will show that, as long as some basic mathematical properties are enforced, the algorithm effectively solves different kinds of integer programming problems in short computational time. Furthermore, we will briefly illustrate some desirable convergence properties of the proposed scheme.

The paper has the following structure: Section 2 presents the mathematical model for the problem. Section 3 provides some insight about how Cross Entropy can be used to solve combinatorial optimization problems. Section 4 introduces the proposed algorithm in its general form. Next, Section 5 presents some interesting results about the convergence property of the algorithm. Sections 6 and 7 illustrate how the general algorithm of Section 4 can be used to tackle two important optimization problems within the reliability realm. We will present computational results to prove the effectiveness of the proposed algorithm. Finally, Section 8 concludes with some interesting remarks.

2 Problem Formulation and Mathematical Properties

Let us consider the general integer program:

$$
\text{(IP):}\quad \left|\begin{array}{lr} \max & z = S(\mathbf{x}) \\ \text{s.t.} & \mathbf{g}(\mathbf{x}) \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}^n \end{array}\right.
$$

where $S : \mathbb{Z}^n \rightarrow \mathbb{R}^+$ represents the objective function, a mapping of $\mathbb{Z}^n$ into $\mathbb{R}^+$, $\mathbf{g} = (g_1, g_2, \ldots, g_K)$ is a set of K knapsack-type constraints, with $g_l : \mathbb{Z}^n \rightarrow \mathbb{R}^+$, and $\mathbf{b} \in \mathbb{R}^n_+$ is the resource availability vector. It is worth noting that neither S nor g_l, $l = 1, \ldots, K$, are required to be linear. Finally, the variable vector $\mathbf{x}$ is an integer vector of size n. In the following, let $\mathcal{X}$ indicate the feasible space of (IP), which is, $\mathcal{X} = \{\mathbf{x} \in \mathbb{Z}^n : \mathbf{g}(\mathbf{x}) \leq \mathbf{b}\}$.

The purpose of the paper is to present a new robust algorithm for the general IP problem with knapsack constraints and especially suited for very large scale instances. The underlying assumption required in order for the algorithm to find good quality solutions is that the objective function S as well as the constraint functions $g_l, l = 1, \ldots, K$, satisfy the "monotonic property" presented below in Definition 1.

We now introduce the notion of monotonic property for a boolean function (See Crama [17]). Let $\mathbb{B}^n$ denote the $n-$dimensional hypercube $\{0, 1\}^n$, let $f(x_1, \ldots, x_n)$ be a boolean function defined on $\mathbb{B}^n$, and let $N = \{1, 2, \ldots, n\}$. The first derivative of f with respect to x_i is the boolean function defined as:

$$\delta_i f(\mathbf{x}) = f_+(\mathbf{x}) - f_0(\mathbf{x}) \tag{1}$$

where $f_+(\mathbf{x}) = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$, $f_0(\mathbf{x}) = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$, for all $\mathbf{x} \in \mathbb{B}^n$.

Definition 1 *A boolean function f is monotonic if, for each $i \in N$, $\delta_i f$ has constant sign on $\mathbb{B}^n$.*

It is a well known fact that any separable function in integer variables can be transformed to an equivalent 0-1 function (See Hammer and Rudeanu [18]). For this reason, the above definition applies to any separable integer function and it is valid for all the problems presented in this paper.

3 Cross-entropy and Combinatorial Optimization

The most important features of CE have been thoroughly exposed in de Boer et al. [12]. For this reason, in this paper we only present those features of CE that are relevant to the problems hereby discussed.

Let us consider a general 0-1 integer maximization problem

$$\text{(P): } z^* = \max_{\mathbf{x} \in \mathcal{X}} S(\mathbf{x}),$$

where $\mathcal{X} \subset \mathbb{B}^n$ represents the feasible region. The CE method associates a stochastic estimation problem to (P) aimed at estimating

$$\mathbb{P}(S(\mathbf{x}) \geq z).$$

With this aim, let us define a family of density functions f on $\mathcal{X}$, parametrized by a vector $\mathbf{p}$. For example, let us assume that we can define a family of Bernoulli density functions:

$$f(\mathbf{x}, \mathbf{u}) = \prod_{i=1}^{n} (u_i)^{x_i} (1 - u_i)^{1-x_i} \tag{2}$$

Consequently, the associated stochastic estimation problem is

$$\text{(EP): } \mathbb{P}(S(\mathbf{x}) \geq z) = \sum_{\mathbf{x} \in \mathcal{X}} I_{\{S(\mathbf{x}) \geq z\}} f(\mathbf{x}, \mathbf{u}),$$

where $I_{\{S(\mathbf{x}) \geq z\}}$ is the indicator function, whose value is 1 if $S(\mathbf{x}) \geq z$ and 0 otherwise.

We want to estimate $l = \mathbb{P}(S(\mathbf{x}) \geq z)$ for $z = z^*$. When the size of $\mathcal{X}$ is large enough, simulation is an acceptable approach to find $\mathbf{x}^*$ such that $z^* = S(\mathbf{x}^*) \geq S(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$. We could estimate l via Monte Carlo simulation by drawing a random sample $\mathbf{X}_1, \ldots, \mathbf{X}_N$ from $f(\mathbf{x}, \mathbf{u})$ with

$$\frac{1}{N} \sum_{i=1}^{N} I_{\{S(\mathbf{X}_i) \geq z\}}.$$

However, the event $\mathbf{X}_i = \mathbf{x}^*$ is a rare event, with probability $1/|\mathcal{X}|$. Hence, the size of the sample might be extremely large. Alternatively, it is possible to estimate l via Importance Sampling (IS). Let us take a random sample $\mathbf{X}_1, \ldots, \mathbf{X}_N$ from a different density function $f(\mathbf{x}, \mathbf{p})$. In this case, we can use the likelihood ratio estimator

$$\hat{l} = \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\mathbf{X}_i) \geq z\}} \frac{f(\mathbf{x}, \mathbf{u})}{f(\mathbf{x}, \mathbf{p})},$$

where $f(\mathbf{x}, \mathbf{p})$ constitutes a change of measure and is chosen such that the cross entropy between $f(\mathbf{x}, \mathbf{p})$ and $f(\mathbf{x}, \mathbf{u})$ is minimal. This corresponds to solving the problem

$$(\mathrm{SP}): \max_{\mathbf{p}} \hat{D}(\mathbf{p}) = \max_{\mathbf{p}} \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\mathbf{X}_i) \geq z\}} \ln f(\mathbf{x}, \mathbf{p}),$$

where $f(\mathbf{x}, \mathbf{p})$ is given by (2). To find the maximum of (SP), we set $\partial \hat{D}(\mathbf{p})/\partial \mathbf{p} = 0$. Consequently, we get the optimal updating rule:

$$\hat{p}_i = \frac{\sum_{i=1}^{N} I_{\{S(\mathbf{X}_i) \geq z\}} x_i}{\sum_{i=1}^{N} I_{\{S(\mathbf{X}_i) \geq z\}}}, \quad i = 1, \ldots, n. \tag{3}$$

Rule (3) is iteratively used with the aim of generating a sequence of increasing threshold values $z^0, z^1, \ldots$, converging either to the global optimum z^* or to a value close to it. At each iteration t, the new value of z^t is used to generate a better vector $\mathbf{p}^t$. The new vector $\mathbf{p}^t$ is, in turn, used to draw a new, hopefully better, sample population, which will lead to a better value of z. The process stops when either we reach the global optimum value z^* or the vector $\mathbf{p}$ converges to a vector in $\mathcal{X}$.

When using rule (3) to generate a random population, however, we find that, depending on the structure of the feasible region of the problem, many sample points might be infeasible. The issue becomes more delicate when dealing with a non-convex feasible region. We investigated different approaches aimed at coping with the presence of several constraints. We now briefly present three possible schemes that allow to encompass feasibility in generating sample points, along with advantages and drawbacks of each one of them.

3.1 Acceptance–Rejection Approach

The Acceptance-Rejection approach analyzes each random generated vector $\mathbf{X}_i$ and checks its feasibility. The vector is accepted if feasible, otherwise it is rejected and a new sample vector $\mathbf{X}_i$ is drawn.

The main advantage of the method is its simplicity, along with the fact that the vector $\mathbf{p}$ need not be modified. Furthermore, no manipulation of the random vector $\mathbf{X}_i$ is required. However, one mayor drawback of the method is that, depending on the structure of the feasible region, the majority of the sample vectors $\mathbf{X}_i$ might be infeasible and, hence, rejected.

3.2 Penalty Approach

The penalty approach is based upon the idea of relaxing the constraints, in a similar fashion to what is done with Lagrangian Relaxation. For example, with respect to the problem (IP), the penalty method would work on the relaxed problem

$$(\text{IPL}) : \ \max \ L(\mathbf{x}, \lambda) = \{S(\mathbf{x}) + \lambda\,(\mathbf{b} - \mathbf{g}(\mathbf{x})) : \lambda \geq 0, \ \mathbf{x} \in \mathbb{B}^n\}\,.$$

However, when implemented, the method proved to be very sensitive to the value of the penalty vector λ. Furthermore, owing to the complex functional structure of S, deriving an updating rule for the penalty values is not a trivial task.

3.3 Projection Approach

We devised a simple mechanism to overcome the infeasibility problem. This method, called *projection* method, exploits the monotonic property of the constraints of (IP). The basic idea is to apply a modification scheme to the random vector $\mathbf{X}_i$ such that the final result is a feasible vector $\mathbf{X}_i^p$.

Let us consider the following definition:

Definition 2 *Given a vector* $\mathbf{x} \in \mathbb{B}^n$ *and a subspace* $\mathcal{X} = \{\mathbf{x} \in \mathbb{B}^n : \mathbf{g}(\mathbf{x}) \leq \mathbf{b}\}$*, a projection of* $\mathbf{x}$ *on* $\mathcal{X}$ *is a vector* $\mathbf{x}^p \in \mathcal{P}_{\mathcal{X}}(\mathbf{x})$*, where:*

$$\mathcal{P}_{\mathcal{X}}(\mathbf{x}) = \left\{\mathbf{x}^p \in \mathcal{X} : x_j - x_j^p \geq 0, \ j = 1, \ldots, n\right\}$$

The projection method works as follow: draw a random vector $\sim \mathsf{Ber}\,(\mathbf{p})$, say $\mathbf{X}_i$; if $\mathbf{X}_i \in \mathcal{X}$, add $\mathbf{X}_i$ to the population, otherwise project $\mathbf{X}_i$ into $\mathcal{X}$ and get $\mathbf{X}_i^p \in \mathcal{X}$. Add $\mathbf{X}_i^p$ to the sample population.

In Section 6 we will show how the projection method can be easily implemented, provided that the monotonic property of Definition 1 is satisfied by all the constraints.

4 General Algorithm

The general algorithm can be applied to discrete variable problems as well as to binary problems. We only present the scheme used with binary problems, since it is always possible to "binarize" a discrete variable problem. More details about the conversion are given in Section 7.

The main algorithm is made up by three phases: (1) draw a random sample population $\sim \mathsf{Ber}\,(\mathbf{p})$; (2) project the infeasible vectors into the feasible region $\mathcal{X}$; and (3) update density parameters via Rule 3. Steps (1)–(3) are repeated until a stopping criterion is met.

Algorithm 1 provides a description of the proposed scheme.

Remark 1. The updating formula (4) is derived from (3), which minimizes the cross entropy distance.

Remark 2. Owing to the projection scheme, the vector $\mathbf{p}$ converges to a feasible point, which is, $\lim_{t\to\infty} \mathbb{P}(\mathbf{p}^t \in \mathcal{X}) = 1$. Consequently, the algorithm always terminates within a finite number of iterations (see Theorem 1).

Remark 3. We clearly establish a direct correspondence between a vector in the binary feasible space $\mathbf{x}^b \in \mathcal{X}$ and a vector $\mathbf{x} \in \mathcal{S}$, feasible to the (IP) problem. We can always transform a binary vector $\mathbf{x}^b$ to its corresponding discrete vector in $\mathcal{O}(n \log_2 \mathrm{ub}_j)$, where ub_j is the upper bound of the j^{th} component of $\mathbf{x}^b$.

Algorithm 1 Cross Entropy (CE)

```
 1: if the problem is not *binary*, transform the problem in binary format – $\mathcal{O}(n \log_2 \mathrm{ub}_j)$.
 2: Set $t := 0$. Generate initial Bernoulli probabilities $\mathbf{p}^t \in (0, 1)$. – $\mathcal{O}(n)$.
 3: draw a sample population $\mathbf{X}_1, \ldots, \mathbf{X}_N \sim \mathsf{Ber}\,(\mathbf{p}^t)$ – $\mathcal{O}(nN)$.
 4: project each infeasible point into the feasible region $\mathcal{X}$ – $\mathcal{O}(nN)$.
 5: sort sample population in ascending order w.r.t. objective function value – $\mathcal{O}(N \log N)$.
 6: select best $\rho N$ points of sample population
 7: compute $\mathbf{p}_j^{t+1}$ as $(\mathcal{O}(\rho N))$:
        $$p_j^{t+1} = \frac{\sum_{i=\lceil \rho N \rceil}^{N} x_{ij}}{\lceil \rho N \rceil}, \; j = 1, \ldots, n \qquad (4)$$
 8: **if** $\mathbf{p}^t \notin \mathbb{B}^n$ **then**
 9:     $t \leftarrow t + 1$ and go back to step 3.
10: **end if**
11: transform the problem to the original format and STOP – $\mathcal{O}(n \log_2 \mathrm{ub}_j)$.
```

5 Convergence

Let us suppose that $|\rho N| = 1$, that is, we use the best point of each population to compute the new probabilities. We want to find out under which conditions the algorithm converges to a point $\mathbf{x}^* \in \mathbb{X}$.

For a Bernoulli, the density function $f(\mathbf{x})$ is :

$$f(\mathbf{x}) = \prod_{i=1}^{n} p_i^{x_i}(1-p_i)^{1-x_i}, \quad x_i \in \{0,1\}, \quad i = 1,\ldots,n$$

Theorem 1 *The CE algorithm converges to a point* $\mathbf{x} \in \mathbf{X}$ *with probability 1.*

PROOF. To prove that the algorithm converges to a point $\mathbf{x}^* \in \mathbb{X}$, it suffices to prove that

$$\lim_{T\to\infty} \mathbb{P}(\mathbf{x}_t \neq \mathbf{x}^*, t = 1,\ldots,T) = 0, \tag{5}$$

where t denotes the current iteration. Let us call $\alpha_t = 1 - \beta_t$; hence, at each iteration $t, (t \geq 1)$, we have:

$$p_{t+1,i} = \begin{cases} \alpha_t p_{t,i}, & \text{if } x_i = 0, \\ \alpha_t p_{t,i} + \beta_1, & \text{if } x_i = 1. \end{cases}$$

In general, we can write:

$$p_{t,i} = p_{1,i}\alpha_1\alpha_2\ldots\alpha_{t-1} + c_{1,i} + c_{2,i} + \ldots + c_{t-1,i},$$

where $c_{t,i}$ indicates whether the β_t terms are present or not. The general expression for $c_{t,i}$ is a combination of $\alpha \times \beta$, depending on the value of x_i at each iteration. In any case, all the terms $c_{t,i} \geq 0$, thus we can write:

$$p_{t,i} \geq p_{1,i} \prod_{t=1}^{T-1} \alpha_t$$

Now, let us suppose that:

$$\prod_{t=1}^{T-1} \alpha_t \geq \frac{1}{t} \tag{6}$$

In this case, we have:

$$p_{t,i} \geq \frac{p_{1,i}}{t} \Rightarrow f_t(\mathbf{x}) \geq \frac{f_1(\mathbf{x})}{t}$$

Now, we can rewrite (5) as:

$$\begin{aligned}\mathbb{P}(\mathbf{x}_t \neq \mathbf{x}^*, t = 1, \ldots, T) &= \prod_{t=1}^{T}(1 - f_t(\mathbf{x})) \leq \prod_{t=1}^{T}(1 - f_1(\mathbf{x})t^{-1}) \\ &\leq \prod_{t=1}^{T} e^{-f_1(\mathbf{x})t^{-1}} = e^{-f_1(\mathbf{x})\sum_{t=1}^{T} t^{-1}}\end{aligned}$$

We conclude that:

$$\lim_{T\to\infty} \mathbb{P}(\mathbf{x}_t \neq \mathbf{x}^*, t = 1, \ldots, T) = \lim_{T\to\infty} e^{-f_1(\mathbf{x})\sum_{t=1}^{T} t^{-1}} = 0$$

Hence the algorithm converges as long as (6) is satisfied.

Corollary 1 *Sufficient conditions for (6) to be true are:*

$$\alpha_{t-1} \geq \left(\frac{1}{t}\right)^{(1-T)}, \ t = 1, \ldots, T$$

PROOF. Considering $\alpha_T = \alpha_{T-1} = \ldots = \alpha_1$, we have:

$$\alpha^{(T-1)} \geq 1/t \Rightarrow \alpha \geq \left(\frac{1}{t}\right)^{(1-T)}$$

which is:

$$\alpha_{t-1} \geq \left(\frac{1}{t}\right)^{(1-T)}, \ t = 1, \ldots, T.$$

6 Software Reliability Problem

In this section we present the specialization of the general CE-based algorithm for the software reliability problem. The section is organized as follows: in the next subsection, we offer a brief introduction to the problem, along with some bibliographical references for the interested reader. Next, in Section 6.2, we present the specialization of the algorithm for the software reliability problem. Finally, in Section 6.3 we illustrate the effectiveness of the algorithm by presenting some results on both benchmark problems drawn from the literature, and random generated very large scale instances.

6.1 Problem Formulation

The software system reliability problem is aimed at identifying the system configuration that maximizes the overall reliability through redundancy allocation, while taking into account a set of resource constraints. Forty percent of software development cost is spent on testing to remove errors and to ensure high quality [19]. One traditional way to increase the reliability of a software system is by increasing the time of debugging. However, due to the high debugging costs, alternative solutions are pursued, like the use of redundant software modules. In software development, redundancies are programs developed by different teams based on the same specifications. Often, in order to ensure the independence of the different modules, these programs are written using different languages, platforms, development methodologies and testing strategies.

The general software system studied is made up by several programs, each performing a different function. Each program contains a set of modules. For each module, a number of different versions exist, each characterized by a cost and a reliability level. The objective is to maximize reliability by choosing the optimal set of versions for each module, allowing redundancy.

The mathematical formulation of a software reliability problem with m modules (each one performing a different function), m_i versions for each module i ($\sum_{i=1}^{m} m_i = n$) and K resource constraints resembles the one of (IP) of Section 2 and is provided below:

$$
\text{(RP):}\quad \left|\begin{array}{lll}
\max & R_s = S(\mathbf{x}) & \\
\text{s.t.} & & \mathbf{g}(\mathbf{x}) \leq \mathbf{b} \\
& & \mathbf{c}^i\mathbf{x} \geq 1,\ \forall i \in M \\
& & \mathbf{x} \in \mathbb{B}^n
\end{array}\right.
$$

where $S : \mathbb{B}^n \to \mathbb{R}^+$, $\mathbf{g} = (g_1, g_2, \dots, g_K)$ represents K knapsack-type constraints, with $g_l : \mathbb{B}^n \to \mathbb{R}^+$, $\mathbf{b}$ is the resource availability vector, $\mathbf{c}^i$ is a vector $(1 \times n)$ whose component j is 1 if x_j belongs to module i, and 0 vice versa. Finally, the variable vector $\mathbf{x}$ is a binary vector, where a value 1 indicates that version j is in the system and 0 vice versa. Note that the monotonic assumption is realistic for this class of problems.

The first set of constraints, $\mathbf{g}(\mathbf{x}) \leq \mathbf{b}$, represents economical and financial constraints (not necessarily linear), while the second set of constraints, $\mathbf{c}^i\mathbf{x} \geq 1$, implies that each module i must have at least one version.

A number of exact and heuristic-based approaches have been proposed in recent years, each one of them addressing a specific class of problems. What emerges from the analysis of the literature is that existing models and algorithms can be used only under specific circumstances. Some of the most relevant approaches are those proposed by Belli and Jedrzejowicz [20], Berman and Ashrafi [21], Shi [22], and, more recently, by Kim and Yum [23], Ravi et al. [24], Kohda and Inoue [25].

6.2 CE-based Algorithm

The algorithm presented in this section is a specification of the CE algorithm of Section 4. We will illustrate two of the most important features of the algorithm, namely, the normalization process, used to guarantee the satisfaction of the second set of constraints of (RP), and the projection scheme adapted to the problem, which ensures that the knapsack-type constraints are satisfied.

In order to ensure that each module has at least one version in the final solution, at each iteration of the algorithm we apply the normalization scheme. Let us suppose that we randomly generate the sample point $\mathbf{X}_i \sim \mathsf{Ber}\,(\mathbf{p})$. Let us assume that module i has m_i versions, indicated by $j = 1, \ldots, m_i$. After generating a random value for version $x_j,\ j \in [1, m_i)$, we call the following scheme:

Procedure 2 Normalization

```
1: if $\sum_{k=1}^{j} x_k = 0$ then
2:     compute $n_e = |\{x_k : p_k^t > 0,\ k = j+1, \ldots, m_i\}|$ remaining versions
3:     $p_k^t \leftarrow \max\left\{p_k^t, \frac{1}{n_e}\right\}, \quad k = j+1, \ldots, m_i$   normalize probabilities
4: end if
```

Beside ensuring that each module has at least one version in the final solution, we need to ensure that the knapsack type constrains are satisfied. This is accomplished by means of a projection scheme. This scheme takes as input an infeasible vector and gradually reduces the number of versions in the solution until a feasible solution $\mathbf{X}_i^p \in \mathcal{P}_{\mathcal{X}}(\mathbf{X}_i)$ is reached. The projection scheme iteratively set to zero the version x_r chosen as

$$r = \underset{\substack{w=1,\ldots,n \\ x_w=1}}{\mathrm{argmax}}\left\{\frac{\Delta\mathrm{g}}{\delta_w H(\mathbf{x})}\right\},$$

where Δ_g is the average amount of resources released when setting x_w to zero, and $\delta_w H(\mathbf{x})$ is the variation in the objective function value. It is easy to see that we iteratively set to 0 the variable that generates the minimum decrease of the objective function per unit of resource released.

We now illustrate the overall algorithm for the software system reliability problem. To simplify the presentation, we introduce some notation. Let $\mathbf{p}_i = \left(\frac{1}{m_i}, \ldots, \frac{1}{m_i}\right)$ be a vector of m_i components that represents the initial probabilities on each module. The termination criterion for the algorithm is given by the maximum number of iterations allowed. Algorithm 3 presents the proposed scheme.

Algorithm 3 Algorithm Cross Entropy Software Reliability Optimization

```
 1: define $\mathbf{p} = (\mathbf{p_1}, \ldots, \mathbf{p_m})$                    initial probability vector
 2: while !MAX_NUMBER_ITERATIONS do
 3:    draw $N = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ vectors with prob $\mathbf{p}$    success prob: $x_j = 1$
 4:    for each module $i = 1, \ldots, m$ do
 5:       for $j = 1, \ldots, m_i$ do
 6:          if $x_{ij} = 0$ then
 7:             normalize $p_{ij+1}, \ldots, p_{im_i}$ s.t. $\sum_{k=1}^{m_i} p_{ik} = \sum_{k=j+1}^{m_i} p_{ik}$    $\mathbf{c}^i\mathbf{x} \geq 1$
 8:             $p_{ik} = \max\{p_{ik}^{old}, p_{ik}^{norm}\},\ k = j+1, \ldots, m_i$    update $\mathbf{p}$
 9:          end if
10:       end for
11:    end for
12:    if $\mathbf{x}_i$ not feasible then
13:       project $\mathbf{x}_i$ into feasible space
14:    end if
15:    sort $\{\mathbf{x}_l\}$ according to $H(x)$                $H_{(1)} \leq H_{(2)} \leq \ldots \leq H_{(N)}$
16:    define quantile $q = \lceil (1-\rho)N \rceil$
17:    define $\mathcal{Q} = \{\mathbf{x}_{(l)} : q \leq l \leq N\}$          set of best $\rho$% points of N
18:    compute $\hat{p}_{ij} = \frac{\sum_{\mathbf{x} \in \mathcal{Q}} x_{ij}}{|\mathcal{Q}|},\ i = 1, \ldots, m;\ j = 1, \ldots, m_i$    update probs
19: end while
```

6.3 Results

In this section we present the results of the algorithm on some problems drawn from the literature on software optimization as well as on some random generated problems of bigger size. The benchmarking tests have been carried out on a Pentium M Centrino© workstation at 1.6 GHz with 512MB of RAM. The CE algorithm has been implemented and compiled using Microsoft C#.NET 2003.

Berman and Ashrafi [21] present four different models, corresponding to four system configurations. We solved problems P1 – P4 with the proposed algorithm and the algorithm always found the global optimal solution in less than 0.01 CPU seconds. However, we provide an iteration-by-iteration analysis of the algorithm behavior when solving problem P2. Table 1 reports the problem parameters, as in Berman and Ashrafi [21]. The system is made up by three modules arranged in series, and eight versions to be distributed among the modules. There is only one knapsack-type constraint, with a total budget of 10.

	Module 1		Module 2		Module 3	
Version	r_j	c_j	r_j	c_j	r_j	c_j
1	0.90	3	0.95	3	0.98	3
2	0.80	1	0.80	2	0.94	2
3	0.85	2	0.70	1	–	–

Table 1
Reliability and Cost for the Berman and Ashrafi [21] Problem

Iteration	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
0	0.33	0.33	0.33	0.33	0.33	0.33	0.50	0.50
1	0.20	1.00	0.80	1.00	0.00	0.80	1.00	0.00
2	0.00	1.00	1.00	1.00	0.00	1.00	1.00	0.00

Table 2
Iterations of the Algorithm on Berman and Ashrafi Problem

Table 2 reports the results of the CE algorithm, using N = 50 and $\rho = 0.1$. The final solution $\mathbf{p}_2$ corresponds to the global optimal solution $x_2 = x_3 = x_4 = x_6 = x_7 = 1$, with a reliability value of 0.9363. Computational time is less than 0.01 CPU seconds.

To test problem P1, we force the algorithm to include exactly one version per module. The algorithm finds the global optimum solution $x_2 = x_4 = x_8 = 1$ with a reliability value of 0.7144 in less than 0.01 CPU seconds.

Berman and Ashrafi [21] also presents two more cases where the objective function presents a slight variation to the original one. These models deal with several programs each performing a specific function. Table 3 presents the problem parameters for this model as in Berman and Ashrafi [21].

With a total budget of 6, the algorithm achieves the global optimum in less than 0.01 CPU seconds with a total reliability of 0.7470 when redundancy is not allowed. The optimal solution is $x_1 = x_4 = x_6 = 1$. If the budget is increased to 10, the optimal solution is then $x_2 = x_4 = x_5 = 1$, also achieved in less than 0.01 CPU seconds with a total reliability of 0.7972. If redundancy

	Module 1		Module 2		Module 3	
Version	r_j	c_j	r_j	c_j	r_j	c_j
1	0.80	2	0.70	1	0.95	4
2	0.85	3	0.90	3	0.90	3

Table 3
Reliability and Cost for the Berman and Ashrafi [21] Problem P3 with $K = 2$ functions and frequencies $F_1 = 0.70$ and $F_2 = 0.30$.

is allowed for this model, the cross entropy method also finds the optimal solution in less than 0.01 CPU seconds.

In order to further test the effectiveness of the algorithm, with special emphasis on very large instances of the reliability problem, we devised a mechanism aimed at generating random instances of the general reliability problem. More details about the random scheme are provided in Caserta and Márquez [26]. Along with the random generation process, we implemented a simple code to solve the software reliability problem via enumeration. The enumeration scheme has been used to test the quality of the solution provided by the proposed algorithm in terms of closeness to the global optimum.

To perform these tests, we have chosen the original version of the algorithm, where only one function is performed within the software system and redundancy is allowed. The random instances generated are of medium and big size. This is, from 5 modules with a maximum of 4 versions per module, to 10 modules with a maximum of 10 versions per module. We named these instances R1, R2 and R3. To achieve the global optimum, we first solved the three problems via enumeration and then compared the results of the algorithm with the global optimum. The parameters of the algorithm for these instances are the same as for the previous ones, except for the sample size that we have ample to 100. We start the test with $\rho = 0.1$. With this initial set up, we achieve the optimal solution for the three instances in less than 0.4 CPU seconds.

For instance R1, with 5 modules and a maximum of 4 versions per module, the global optimum is achieved after 3 iterations, in less than 0.03 CPU seconds and a reliability value of 0.8617. The second instance, R2, consists of 7 modules with a maximum of 4 versions per module. For this instance we need 4 iterations to achieve the global optimal value of 0.9131 in less than 0.06 CPU seconds. The last instance we tested consists of 10 modules with a maximum of 10 versions per module. To solve this problem, the CE algorithm needs a total of 9 iterations that runs in less than 0.36 CPU seconds, achieving the global optimum, with a reliability value of 0.9848.

These tests were run with a value of $\rho = 0.1$. However, we tested the behavior of the algorithm with different values of ρ. As shown in Table 4, for values of ρ between 0.1 and 0.8 the optimal value is always achieved. Problems arise when the value of ρ is extreme, that is either bigger than 0.8 or smaller than 0.1.

ρ	R1	R2	R3
0.01	0.8584	0.8809	0.9524
0.05	0.8617	0.9146	0.9759
0.1	0.8617	0.9247	0.9853
0.2	0.8617	0.9247	0.9853
0.3	0.8617	0.9247	0.9853
0.4	0.8617	0.9247	0.9853
0.5	0.8617	0.9247	0.9853
0.6	0.8617	0.9247	0.9853
0.7	0.8617	0.9247	0.9853
0.8	0.8617	0.9247	*0.9852*
0.9	0.8617	0.9247	*0.9849*
0.95	0.8584	*0.9247*	*0.983*
0.99	*0.8617*	*0.901*	*0.9747*

Table 4
Values of the objective function with different ρ

Figures in italic indicate that after 40 iterations convergence was not achieved. For instance R3 with $\rho = 0.8$ we need to increase the maximum number of iterations from the original 40 to 70 to achieve convergence. This is not a drawback, since the algorithm is still very fast, and takes 3.405 CPU seconds to find an optimal solution.

From these results we can conclude that for a value of ρ between 0.1 and 0.8, with a sample size of 100, the algorithm finds the optimal solution in three random instances of considerable size. This confirms the robustness of the CE algorithm for the reliability problem, not only in terms of accuracy but also in terms of computational time.

7 Complex System Reliability Problem

This section offers some indications about how to apply the general CE algorithm to discrete, non-binary, problems. The problem studied is known as complex system reliability optimization. The section is organized as follows: in the next subsection, we introduce the problem. Next, in Section 7.2, we present the specialization of the algorithm for the discrete reliability problem. Finally, in Section 7.3 we offer some results of the algorithm on one benchmark problem from the literature.

7.1 Problem Formulation

The problem studied here is "the optimal redundancy allocation in complex system configurations with multiple knapsack constraints," defined as

$$(\mathrm{P}):\ \max\left\{R_s = S(\mathbf{x}) : \mathbf{g}(\mathbf{x}) \leq \mathbf{b}, \mathbf{x} \in \mathbf{Z}_+^n\right\},$$

where $S : \mathbf{Z}_+^n \to \mathbf{R}_+$, $\mathbf{b} \in \mathbf{R}_+^m$ and $\mathbf{g} = (g_1, g_2, \ldots, g_K)$ are K knapsack-type constraints, with $g_l : \mathbb{Z}^n \to \mathbb{R}^+$. As usual, we assume that S is nondecreasing in $\mathbf{x}$.

Problem (P) is maybe the most challenging among redundancy allocation problems. The major difficulty of solving (P) is associated with its multi-linear objective function, whose specific functional form depends on the particular system configuration and the number of components in it. An excellent overview of different methods proposed to solve complex system reliability problems is given in Kuo and Prasad [27].

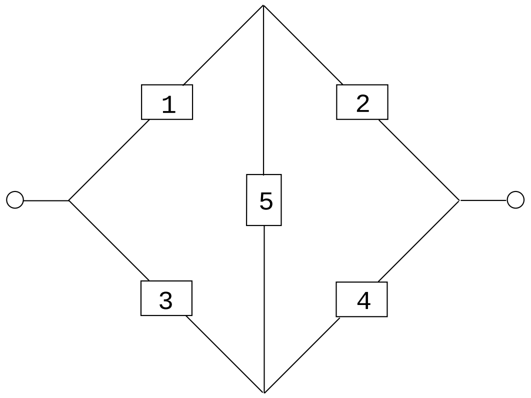

Fig. 1. Complex System Configuration: 5-link System

The objective function value of a complex system is computed via conditional probability. For example, let us consider the complex system configuration of Figure 1. We indicate the reliability of link i as $R_i = 1 - (1 - r_i)^{x_i}$ and

$Q_i = 1 - R_i$, where x_i indicates the number of redundant components of link i ($x_i \geq 1$). Consequently, the overall reliability of the system can be computed as:

$$\begin{aligned} R_s &= R_s|R_5 + R_s|Q_5 \\ &= [(1 - Q_1Q_3)(1 - Q_2Q_4)R_5] + [1 - (1 - R_1R2)(1 - R_3R_4)]\, Q_5. \end{aligned}$$

7.2 CE-based Algorithm

The algorithm for the discrete case differs from the one presented in Section 6 in that we need to "binarize" the problem before applying CE. In the following, let us indicate with $\mathbf{x}^j = \mathbf{e}_j$, a vector with all zeros except in position j, where a 1 is present. We transform a discrete variable x_j into a set of bits using the following scheme:

Procedure 4 Transform-to-Binary

```
1: set $\mathbf{x} = 1$                                        each variable at its lower bound
2: for each variable $x_j$ in the problem do
3:     $\mathrm{ub}_j = \min_{l=1,\ldots,K} \left\{ \frac{b_l - g_l(\mathbf{x})}{g_l(\mathbf{x}^j)} \right\}$      minimum over all constraints
4:     $\mathrm{nBits}_j = \lceil \log_2 ub_j + 1 \rceil$          set number of bits for variable $x_j$
5: end for
```

7.3 Results

The first test problem is from Aggarwal [28] and is a small 5-link bridge network system with a budget of 20 units, as represented in Figure 1. Table 5 presents the problem parameters. The global optimal solution is found at $\mathbf{x}^* = [3\,1\,2\,2\,1]$ with $R_s^* = 0.9932$. The proposed algorithm finds the global optimum in four iterations in less than 0.01 CPU seconds.

j	1	2	3	4	5
r_j	0.70	0.80	0.75	0.85	0.90
a_j	2	2	1	3	3

Table 5
Parameters of Aggarwal's problem

The second test problem is a small 4-link composite system with two constraints from Shi [22]. Constraint 1 has a budget of 30 units while constraint 2 has a budget of 40 units. See Table 6 for the problem parameters and Figure 2 for the system configuration. Despite its small size, this is a difficult problem to solve with the global optimum of $R_s^* = 0.9974$ hidden at $\mathbf{x}^* = [3\,1\,1\,1]$. The algorithm finds the global optimal solution within 5 iterations in less than 0.01 CPU time. In Table 7, we illustrate the trajectory of the algorithm through the values of the vector $\mathbf{p}$. The parameters of the algorithm are set to N = 50 and $\rho = 0.1$.

j	1	2	3	4
r_j	0.80	0.75	0.70	0.65
a_{1j}	6	4	3	2
a_{2j}	9	4	4	3

Table 6
Parameters of Shi's problem

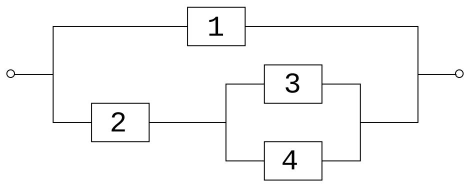

Fig. 2. 4-link, 2-constraint composite system from Shi

Iteration	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}
0	0.50	0.50	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33
1	0.50	0.50	0.00	1.00	0.50	0.00	0.70	0.30	0.10	0.70	0.80
2	0.50	0.60	0.00	0.90	0.60	0.00	0.40	0.60	0.10	0.80	0.90
3	1.00	0.30	0.00	0.70	0.30	0.00	0.50	0.50	0.00	0.70	0.50
4	1.00	0.70	0.00	0.30	0.70	0.00	0.30	0.70	0.00	0.30	0.70
5	1.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00
$\mathbf{x}^*$	3			1			1			1	

Table 7
Iterations of the Algorithm on Shi's Problem

8 Conclusion

We have presented a new, cross entropy based, algorithm for a wide set of combinatorial optimization problems. We exploit a basic mathematical property of the problem, which is, the objective function as well as the constraint functions are non-decreasing in $\mathbf{x}$. The proposed scheme provides some insight with respect to how to deal with knapsack-type constraints and presents convergence properties that guarantee the termination of the algorithm itself.

We tested the robustness of the algorithm with two different classes of problems, both within the reliability realm. In both cases, the algorithm proved to be very effective, in terms of solution quality as well as computational time. The proposed algorithm is especially suited for very large instances, when exact approaches are doomed to fail.

References

[1] M. R. Garey and D. S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness.* Freeman, 1979.

[2] T. A. Feo and G. C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[3] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, 1997.

[4] M. Dorigo and G. Di Caro. *The Ant Colony Optimization Meta-heuristic.* In M. Dorigo, D. Corne and F. Glover - New Ideas in Optmization, pg. 11-32 McGraw-Hill, Cambridge, MA, 1999.

[5] M. Dorigo and G. Di Caro. *New Ideas in Optmization*, chapter The Ant Colony Optimization Meta-heuristic, pages 11–32. McGraw-Hill, Cambridge, MA, 1999.

[6] C. Blum and A. Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3):268–308, September 2003.

[7] R. Y. Rubinstein. Optimization of Computer Simulation Models with Rare Events. *European Journal of Operations Research*, 99:89–112, 1997.

[8] R. Y. Rubinstein. The Simulated Entropy Method for Combinatorial and Continuous Optimization. *Methodology and Computing in Applied Probability*, 2:127–190, 1999.

[9] R. Y. Rubinstein. *Stochastic Optimization: Algorithms and Applications*, chapter “Combinatorial Optimization, Cross-entropy, Ants and Rare Events”, pages 304–358. Kluwer, S. Uryasev and P. M. Pardalos edition, 2001.

[10] R. Y. Rubinstein. The Cross-Entropy Method for Combinatorial and Continuous Optimization. *Methodology and Computing in Applied Probability*, 2:127–190, 1999.

[11] R. Y. Rubinstein and D. P. Kroese. *The Cross-Entropy Method: a Unified Approach to Combinatorial Optimization, Monte Carlo Simulation, and Machine Learning.* Springer-Verlag, 2004.

[12] P. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A Tutorial on the Cross-Entropy Method. *Annals of Operations Research*, 134(1):19–67, January 2005.

[13] G. Alon, D. Kroese, T. Raviv, and R. Y. Rubinstein. Application of the Cross-Entropy Method to the Buffer Allocation Problem in a Simulation-Based Environment. *Annals of Operations Research*, 1(134):137–151, January 2005.

[14] K. Chepuri and T. Homem de Mello. Solving the Vehicle Routing Problem with Stochastic Demands using the Cross-Entropy Method. *Annals of Operations Research*, 1(134):153–181, January 2005.

[15] R. Y. Rubinstein. The cross-entropy method and rare-events for maximal cut and bipartition problems. *ACM Transactions on Modelling and Computer Simulation*, 12(1):27–53, 2002.

[16] J. Keith and D. P. Kroese. SABRES: Sequence Alignment by Rare Event Simulation. In *Proceedings of the 2002 Winter Simulation Conference, San Diego*, pages 320–327, 2002.

[17] Y. Crama. Recognition Problems for Special Classes of Polynomials in 0-1 Variables. *Mathematical Programming*, 44:139–155, 1987.

[18] P. L. Hammer and S. Rudeanu. *Boolean Methods in Operations Research and Related Areas*. Springer-Verlag, Berlin, New York, 1968.

[19] W. Kuo, V. R. Prasad, F. A. Tillman, and C. Hwang. *Optimal Reliability Design*. Cambridge University Press, 2001.

[20] F. Belli and P. Jedrzejowicz. An Approach to the Reliability Optimization of Software with Redundancy. *IEEE Transaction on Software Engineering*, 17(3), March 1991.

[21] O. Berman and N. Ashrafi. Optimization Models for Reliability of Modular Software Systems. *IEEE Transaction on Software Engineering*, 19(11), November 1993.

[22] D. H. Shi. A New Heuristic Algorithm for Constrained Redundancy-Optimization in Complex Systems. *IEEE Transactions on Reliability*, R-36(5):621–623, 1987.

[23] J. H. Kim and B. J. Yum. A Heuristic method for Solving Redundancy Optimization Problem in Complex Systems. *IEEE Transactions on Reliability*, 42(4):572–578, 1993.

[24] V. Ravi, B. S. N. Murty, and P.J. Reddy. Nonequilibrium Simulated Annealing Algorithm Applied to Reliability Optimization of Complex Systems. *IEEE Transactions on Reliability*, 46(2):233–239, 1997.

[25] T. Kohda and K. Inoue. A Reliability Optimization Method for Complex Systems with the Criterion of Local Optimality. *IEEE Transactions on Reliability*, R-31(1):109–111, 1982.

[26] M. Caserta and A. Márquez. Metaheuristic Algorithm for Software System Reliability Problem. *submitted to "IEEE Transaction on Software Engineering*, 2005.

[27] W. Kuo and V. R. Prasad. An Annotated Overview of System-Reliability Optimization. *IEEE Transactions on Reliability*, 49(2):176–187, 2000.

[28] K. K. Aggarwal. Redundancy optimization in general systems. *IEEE Transaction on Reliability*, R-25:330–332, 1976.