Hybridizing the Cross Entropy Method: An Application to the Max-Cut Problem

MANUEL LAGUNA
Leeds School of Business, University of Colorado at Boulder, USA
laguna@colorado.edu

ABRAHAM DUARTE
Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Spain.
Abraham.Duarte@urjc.es

RAFAEL MARTÍ
Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain
Rafael.Marti@uv.es

Abstract

Cross entropy has been recently proposed as a heuristic method for solving combinatorial optimization problems. We briefly review this methodology and then suggest a hybrid version with the goal of improving its performance. In the context of the well-known max-cut problem, we compare an implementation of the original cross entropy method with our proposed version. The suggested changes are not particular to the max-cut problem and could be considered for future applications to other combinatorial optimization problems.

Keywords: combinatorial optimization, metaheuristics, max-cut problem, local search, cross entropy

Version: October 1, 2007

1. Introduction

The cross entropy (CE) method was conceived by Rubinstein (1997) as a way of adaptively estimating probabilities of rare events in complex stochastic networks. The method was soon adapted to tackle combinatorial optimization problems (Rubinstein, 1999 and 2001). Recently, the *Annals of Operations Research* devoted a volume to the cross entropy method (de Boer, et al. 2005a). Applications of the CE method to combinatorial optimization include vehicle routing (Chepuri and Homem-de-Mello 2005), buffer allocation (Alon, et al. 2005), the traveling salesman problem (de Boer, et al. 2005b), and the max-cut problem (Rubinstein, 2002). The description of other applications, a list of references and computer implementations of the CE method can be found at http://www.cemethod.org.

As stated by de Boer, et al. (2005b), the CE method, in its most basic form, is a fairly straightforward procedure that consists of iterating the following two steps:

1. Generate a random sample from a pre-specified probability distribution function
2. Use the sample to modify the parameters of the probability distribution in order to produce a "better" sample in the next iteration

The basic CE method is very easy to implement, particularly when dealing with combinatorial optimization problems for which the natural representation of solutions is a binary string. The methodology is quite intuitive, focusing on the observations with the best objective function values in order to bias the sampling process. In our experience, however, the CE method (as described in the tutorial by Boer, et al. 2005b) has some limitations. Specifically, the method requires of very large samples for finding high-quality solutions to difficult problems and the search parameters are generally difficult to adjust. The large samples make the process slow and the lack of systematic mechanisms for searching the solution space—relying heavily in randomization—make the outcomes unpredictable. This was our motivation for exploring changes and extensions that resulted in a hybrid version of the original CE methodology.

The proposed changes and extensions, which attempt to accelerate the CE method, consist of modifying the way samples are treated in step 2 (above) and adding local search to improve upon the trial solutions that the method generates. We want to point out that it is not our intention to diminish the merit of the original CE method, which was conceived as a generic procedure capable of providing solutions of reasonably good quality to a wide range of optimization problems. Our goal is to suggest a way of hybridizing the original proposal in order to make it more competitive when compared to specialized procedures. We view this process as being similar to the transformations experienced by the GA (Genetic Algorithms) community, which over the years has incorporated elements such as local search and elitism that are departures from the original methodology. The hybrid version of CE that we suggest takes advantage of context information and therefore departs from the original method in that it is no longer a generic procedure. However, we suggest a general way of adding problem context that makes the resulting hybridization applicable to more than one class of combinatorial optimization problems. In this work, we use the max-cut problem for illustration purposes.

2. A Hybridized Cross Entropy Method

In order to facilitate the description of our proposed *hybrid cross entropy* (HCE) method, we focus on unconstrained combinatorial problems with solutions represented as binary strings. Hence, a solution to a problem is given by $\boldsymbol{x} = (x_1, \ldots, x_n)$ where $x_j \in \{0, 1\}$ for all j and all possible instantiations of $\boldsymbol{x}$ are feasible. We assume that the objective is to maximize the value of a real function $f(\boldsymbol{x})$.

As part of the CE methodology, we define a vector $\boldsymbol{p} = (p_1, \ldots, p_n)$ of probabilities corresponding to parameters of n independent Bernoulli random variables. The Bernoulli distribution has been found effective in the context of binary vectors, as indicated by de Boer, et al. (2005b). This means that, in the sampling process, the probability that $x_j = 1$ is a Bernoulli random variable with parameter p_j. In the absence of a priori information, it is customary to initialize the probabilities for the Bernoulli random variables to ½.

The CE method employs five parameters:

N =	sample size
ρ =	cutoff point for high-quality observations
α =	smoothing constant for updating $\boldsymbol{p}$
k =	limit on the number of iterations without improvement
K =	limit on the total number of iterations

Figure 1 shows a pseudo code of the CE method. The procedure starts by initializing the probability vector $\boldsymbol{p}$, the best objective function value $f(\boldsymbol{x}^*)$, and the iteration counters t (total number of iterations) and t' (iterations without improvement). These are steps 1 to 3 in Figure 1.

The main loop in Figure 1 includes the two main tasks that every CE method must perform, namely, the generation of a sample and the updating of the parameters associated with the chosen probability distribution. Step 4 generates a sample of size N using independent Bernoulli trials with probability of success given by $\boldsymbol{p}^t$. The ordering of the sample in step 5 is such that the best observation is placed in the first position of the list and the worst is placed in the last position. The v-values calculated in step 6 are the unadjusted probabilities for the current iteration. They are simply the average of values corresponding to the variables of the top $\lceil \rho N \rceil$ solutions in the current sample. Note that if $\alpha = 1$, the updating of the probability vector in step 7 is such that $\boldsymbol{p}^t = \boldsymbol{v}$. This is the "unsmoothed updating" of the probability vector that is considered one of the possible variants of the CE method. Steps 8 and 9 update the best solution found and reset the counter of the number of iterations without improving, respectively. The global iteration counter is updated in step 10.

```
1.  $p^0 = (½, …, ½)$
2.  $f(x^*) = 0$
3.  $t = 0$ and $t' = 0$
while ($t' < k$ and $t < K$)
{
        4.  Generate a sample of size $N$, where the probability that $x_j^s = 1$
            is $p_j^t$ for $j = 1, …, n$ and $s = 1, …, N$.
        5.  Order the sample in such a way that $f(x^1) \geq f(x^2) \geq … \geq f(x^N)$.
        6.  Calculate $v_j = \frac{\sum_{s=1}^{\lceil \rho N \rceil} x_j^s}{\lceil \rho N \rceil}$ for $j = 1, …, n$.
        7.  Update $p_j^t = \alpha v_j + (1 - \alpha) p_j^{t-1}$ for $j = 1, …, n$.
        if ($f(x^*) \geq f(x^1)$) then $t' = t' + 1$ else
        {
                8.  $x^* = x^1$ and $f(x^*) = f(x^1)$
                9.  $t' = 0$
        }
        10. $t = t + 1$
}
```

Figure 1. Pseudo code for the CE method with sample size controlled by ρ and probability updates smoothed by α

The CE methodology focuses on combinatorial optimization problems that can be represented as weighted graphs and classifies them into two groups: 1) those problems whose decision variables are associated with the edges of the graph (*edge* problems) and 2) those problems whose decisions relate to the nodes of the graph (*node* problems). The traveling salesman problem (TSP) is an example of the former while the max-cut problem is an example of the latter.

The sample size N and the value of ρ are critical to the performance of the CE method. de Boer, et al. (2005b) suggest that, for *edge* problems, the size of the sample should be a function of n^2, for instance, $N = cn^2$, for $c > 1$. This means that for a TSP with $n = 100$ cities, each iteration of the CE method would require a sample larger than 10,000 observations. For *node* problems, the recommendation is to make N a linear function of n, for instance, $N = cn$ for $c > 1$. These recommendations assume that there is only one parameter value that needs to be estimated for the probability distribution associated with each variable in the problem. For example, the Bernoulli distribution has one parameter only (i.e., the probability of success), but other distributions that may be appropriate in some applications (e.g., the Normal or Beta) have more than one parameter.

The CE literature also offers guidelines for adjusting ρ based on the value of n (see e.g. de Boer, et al. 2005b). Generally speaking, it is recommended for the value of ρ to decrease as N increases. For instance, it is recommended to set $\rho = 0.01$ if $n \geq 100$ and make $\rho \approx \ln(n)/n$ when $n < 100$.

Although these guidelines seem intuitive and are based on some experimentation, our experience has been that these critical parameters are difficult to adjust in order to find high-quality solutions and to make the CE method converge. Note that the *K* parameter is added as a safety mechanism, given that, theoretically, the method should converge. Convergence in our context is defined as the point where either $p_j^t \in \{0,1\}$ and/or $f(x^1)$ has not changed in *k* consecutive iterations.

These difficulties were the main motivation for hybridizing CE and developing the version that we are labeling the HCE method. To aid convergence and reduce the sample size, we recommend the use of a local optimization process. The local optimizer is applied to a percentage δ of solutions in the sample. That is, given a sample of size *N*, $\lceil \delta N \rceil$ solutions are subjected to the local optimization in a manner described in Figure 2. The pseudo code in Figure 2 assumes that the sample has been ordered and therefore $\boldsymbol{x}^1$ is the best solution in the sample and $\boldsymbol{x}^N$ is the worst.

The rationale for the structure in Figure 2 is that subjecting the best solution in the sample to the local optimization will tend to accelerate the convergence of the procedure toward solutions of higher quality, especially in the early iterations of the method. Note that, for deterministic local optimization, step 2 in Figure 2 is only executed if the best solution $\boldsymbol{x}^1$ has changed since the last call to the local optimizer. When local optimization includes probabilistic choices, step 2 is applied even if the best solution has not changed since the last execution of the local optimizer. Another interesting aspect of the process in Figure 2 is the local optimization of a diverse set of solutions. This strategy expands the exploration of the solution space and encourages the search to move to regions that contain additional local optima. We have observed that applying the local optimization to the best $\lceil \delta N \rceil$ solutions (instead of the diverse set) drives the method to early convergence and generally worst outcomes.

```
1.  $L = \varnothing$
2.  Locally optimize $\boldsymbol{x}^1$ and add it to $L$
while ($|L| < \lceil \delta N \rceil$)
{
        3.  Find the sample solution $\boldsymbol{x}^s \notin L$ that is the "farthest away"
            from the set $L$ of locally optimal solutions
        4.  Locally optimize $\boldsymbol{x}^s$ and add it to $L$
}
```

Figure 2. Pseudo code for local optimization of sample solutions

The HCE method requires of a distance metric to implement step 3 of Figure 2. That is, it is necessary to define the meaning of a solution being "far away" from a set of other solutions. For a combinatorial problem with solutions represented as binary strings, such as the max-cut problem, we define the distance $d(\boldsymbol{x}, L)$ between a solution $\boldsymbol{x}$ and a set of solutions $\boldsymbol{x}^s \in L$ as follows:

$$d(\boldsymbol{x}, L) = \min_{s=1,\ldots,|L|} \left(\sum_i \text{abs}\left(x_i - x_i^s\right) \right)$$

Then, the solution $\boldsymbol{x}$ that is the farthest away from L is the one that has the maximum $d(\boldsymbol{x}, L)$ value of all the solutions under consideration. Hence $d(\boldsymbol{x}, L)$ is the minimum Hamming distance of $\boldsymbol{x}$ with the vectors belonging to L.

The addition of local search to the CE method has the effect of reducing the size of the sample needed to find high-quality solutions. The HCE method is summarized in Figure 3. The first 5 steps in Figure 3 are very similar to the ones in Figure 1. The main difference is that the stopping criterion has been simplified to performing a total number of iterations K, eliminating the parameter k and the corresponding counter t'. In step 6, the local optimization is applied as described in Figure 2. The updating of the unadjusted probability values (i.e., the $\boldsymbol{v}$ vector) is now done in step 7. We point out that the HCE method employs the entire sample to update $\boldsymbol{v}$. This is equivalent to setting $\rho = 1$ in the CE method. Hence, in the HCE method there is no need for adjusting ρ but now we must adjust the value of δ. Regarding the sample size, we have observed that the relationship $N = cn$ is still valid (for *node* problems). However, the effective values of c are now between 0.01 and 0.05. Steps 8 to 10 in Figure 3 are similar to steps 7 to 10 in Figure 1, with the difference that the HCE method does not keep track of the number of iterations without improvement of the best solution.

```
1.  $\boldsymbol{p}^0 = (½, …, ½)$
2.  $f(\boldsymbol{x}^*) = 0$
3.  $t = 1$
while ($t < K$)
{
        4.  Generate a sample of size $N$, where the probability that $x_j^s = 1$
            is $p_j^t$ for $j = 1, …, n$ and $s = 1, …, N$.
        5.  Order the sample in such a way that $f(\boldsymbol{x}^1) \geq f(\boldsymbol{x}^2) \geq … \geq f(\boldsymbol{x}^N)$.
        6.  Apply the local optimization procedure of Figure 2.
        7.  Calculate $v_j = \frac{\sum_{s=1}^{N} x_j^s}{N}$ for $j = 1, …, n$.
        8.  Update $p_j^t = \alpha v_j + (1-\alpha) p_j^{t-1}$ for $j = 1, …, n$.
        if ($f(\boldsymbol{x}^*) < f(\boldsymbol{x}^1)$)
        {
                9.  $\boldsymbol{x}^* = \boldsymbol{x}^1$ and $f(\boldsymbol{x}^*) = f(\boldsymbol{x}^1)$
        }
        10. $t = t + 1$
}
```

Figure 3. Pseudo code for the HCE method

It might be tempting to argue that the HCE method described in Figure 3 is nothing else but a multi-start procedure with a local search. However, our experiments show that this is far from the truth. In other words, if we eliminate steps 7 and 8, fixing $\boldsymbol{p}^t$ to its initial value, the resulting procedure is significantly inferior to the proposed HCE method. This means that it is not the local search alone but rather the combination of the local search and the sampling procedure that contributes to the quality of the solutions found with the HCE method.

3. The HCE Method for the Max-Cut Problem

The max-cut problem is a well-known NP-hard problem in combinatorial optimization and is well-suited as test case in the current context. Given a graph $G(V, E)$, where $V = \{1, \ldots, n\}$ is the set of vertices and E is the set of edges, the problem consists of finding a partition of the nodes into two subsets V_1 and V_2 such that the sum of the weights of the edges going from one subset to the other is maximized. The problem can be formulated as an integer quadratic program, as shown in Festa, et al. (2002). This formulation leads to solution procedures for the max-cut problem that are based on solving semidefinite programs, as shown by Goemans and Williams (1995). Heuristically, the max-cut problem has been approached by representing a solution as a binary vector $\boldsymbol{x}$, where we can arbitrarily say that $x_i = 1$ means that node i is assigned to V_1. Since the labels for the subsets V_1 and V_2 are arbitrary, one way of avoiding symmetries in the solutions is to assign the first node to the first group and represent all the solutions as $\boldsymbol{x} = (1, x_2, x_3, \ldots, x_n)$ and $x_i \in \{0, 1\}$. Let w_{ij} be the weight of the edge that connects nodes i and j, then the objective function value for a cut vector $\boldsymbol{x}$ is given by:

$$f(\boldsymbol{x}) = \sum_{j>i:x_i \neq x_j} w_{ij}$$

The application of the CE method to the max-cut problem is easily achieved by a direct implementation of the pseudo-code in Figure 1, as done by Rubinstein (2002). For the HCE method, we also need to implement a local optimizer suitable for the max-cut problem. As proposed by Festa, et al. (2002), given a solution $\boldsymbol{x}$, we define two quantities for each node i ($i = 2, \ldots, n$):

$$\sigma_0(i) = \sum_{j:x_j=0} w_{ij} \text{ and } \sigma_1(i) = \sum_{j:x_j=1} w_{ij}$$

Using these quantities, a local neighborhood may be defined as consisting of examining the benefit of changing the assignment of nodes (except node 1) from their current subset to the other. We create a random ordering of all nodes i (for $i = 2, \ldots, n$) and use the ordering to examine, one by one, possible node exchanges. Reassignment of a node occurs according to these rules:

1. **if** $x_i = 0$ and $\sigma_0(i) - \sigma_1(i) > 0$ **then** make $x_i = 1$
2. **if** $x_i = 1$ and $\sigma_1(i) - \sigma_0(i) > 0$ **then** make $x_i = 0$

The process stops when after a complete examination of the list, no node can be reassigned according to these rules. Once a local optimal point has been reached, it does not matter in which order the nodes are examined and therefore we do not apply the local optimization to a locally optimal solution. Such an application would result in a single pass through the nodes that would verify that indeed the current solution is locally optimal with respect to the defined neighborhood.

4. Computational Experiments

We have several goals for our experimentation. First, we want to perform tests with the CE method to fine-tune its parameters and attempt to match the results reported by Rubinstein (2002). We then want to fine-tune the HCE method and apply it to the relatively small instances suggested by Rubinstein (2002). Finally, we want to figure out whether the implementations of the CE and HCE methods are competitive when compared to state-of-the-art procedures specifically designed for the max-cut problem. For our computational experiments we employ the following sets of existing test problem instances.

Set 1 — Rubinstein (2002) introduced six problem instances on artificially created complete graphs with known optimal solutions. The problems have 200 vertices and the weights range from 1 to 5.

Set 2 — This set comes from the 7th DIMACS Implementation Challenge and is publicly available at http://dimacs.rugters.edu/Challenges/Seventh. The number of vertices ranges from 512 to 3375 and the number of edges from 1536 to 10125. Burer et al. (2001) use all four instances in their experiments. Festa et al. (2002) consider only one of them.

Set 3 — Helmberg and Rendl (2000) introduced a set of graphs with several densities and sizes. Burer et al. (2001) and Festa et al. (2002) use the 24 graphs in the set shown in Table 1. They consist of toroidal, planar and random graphs with weights taking the values 1, 0, or -1.

Graph	*n*	Density
G1, G2, G3	800	6.12%
G11, G12, G13	800	0.63%
G14, G15, G16	800	1.58%
G22, G23, G24	2000	1.05%
G32, G33, G34	2000	0.25%
G35, G36, G37	2000	0.64%
G43, G44, G45	1000	2.10%
G48, G49, G50	3000	0.17%

Table 1. Problem instances in Set 3

Set 4 — This set contains twenty instances described in Festa et al. (2002). Ten of the graphs have 1000 vertices and density equal to 0.60% and the other ten have 2744 vertices and density equal to 0.22%. The weight values are either 1, 0, or -1.

All experiments were performed on a personal computer with a 3.2 GHz Intel Xenon processor and 2.0 GB of RAM. In our first experiment we implemented the CE method shown in Figure 1, which corresponds to Algorithm 4.1 in Rubinstein (2002). According to Rubinstein, the best parameter values to solve the problems in Set 1 with the CE method are $\rho = 0.01$, $\alpha = 0.9$ and $k = 5$. In his experiments, he executes the CE method for sample sizes ranging from 200 to 2000. Table I in Rubinstein (2002) shows, for each graph in the test set, the deviation from the optimal solution and the total number of iterations (i.e., the final value of t after the process in Figure 1 stops, which as Rubinstein, we will denote by T). Rubinstein does not specify whether the results are the best from several runs of the CE method or they are average values. When reporting results associated with a method that has as many random components as CE, it is essential to either report results on a very large set of problem instances or, when dealing with a small set (as in the case of Set 1), execute the procedure several times on each available instance. Regardless of how the numbers in Table I of Rubinstein (2002) were obtained, we were unable to match these results and therefore proceeded to fine-tune the parameters of the CE implementation.

We employ the fine-tuning system known as Calibra (Adenso-Diaz and Laguna, 2006) with the goal of finding the most effective values for N, ρ, and α. We consider that $N = cn$ and therefore search for an effective c-value. Calibra utilizes design of experiments and locally searches for the best values for up to 5 input parameters. We configured Calibra to use Set 1 as the training set to search within the following set of values:

$$c = \{1.00, 1.01, 1.02, \ldots, 10.00\}$$
$$\rho = \{0.01, 0.02, 0.03, \ldots, 0.10\}$$
$$\alpha = \{0.5, 0.6, 0.7, \ldots, 1.0\}$$

We did not include k for fine-tuning because the quality of the solutions generated by the CE procedure can only increase with the value of k. Calibra uses solution quality for searching and hence would always choose the largest allowed value for k if this parameter is allowed to be changed during the fine-tuning process. We set k to 10 and K to 1000, letting Calibra adjust the other 3 parameters. We did this because with $k = 5$ we did not find a combination of the other three parameters that would yield solutions of the quality reported by Rubinstein (2002). The Calibra run, with $k = 10$, suggested the following settings: $N = 5.87n$, $\rho = 0.02$ and $\alpha = 1$.

Table 2 reports the results obtained from running our CE implementation with $k = 10$, $\rho = 0.02$ and $\alpha = 1$ on the six problem instances in Set 1. We have empirically found that solution quality is highly dependent on the seed for the random number generator. In order to obtain statistically significant results we ran the method 20 times on each instance. Table 2 presents the relative errors ε_i and the associated number of iterations T_i as a function of the sample size N for the six instances in this set (i.e., for i=1,…,6). Table 3 reports the associated computing times in seconds. Results in Tables 2 and 3 are the average values across the 20 runs.

N	ε_1 (T_1)	ε_2 (T_2)	ε_3 (T_3)	ε_4 (T_4)	ε_5 (T_5)	ε_6 (T_6)
200	0.163 (68.6)	0.142 (70.2)	0.255 (66.3)	0.059 (71.1)	0.317 (21.4)	0.040 (20.8)
400	0.062 (40.6)	0.006 (49.4)	0.081 (38.2)	0.006 (44.9)	0.172 (25.5)	0.025 (26.2)
800	0.000 (25.6)	0.000 (31.3)	0.000 (23.7)	0.000 (27.9)	0.050 (26.1)	0.011 (30.3)
1200	0.000 (21.4)	0.000 (24.7)	0.000 (20.6)	0.000 (22.7)	0.013 (25.3)	0.006 (31.2)
1600	0.000 (21.1)	0.000 (23.0)	0.000 (20.5)	0.000 (22.2)	0.008 (25.1)	0.002 (30.5)
2000	0.000 (20.9)	0.000 (22.5)	0.000 (20.8)	0.000 (21.9)	0.001 (25.4)	0.001 (29.6)

Table 2. CE method results for Set 1

Our first observation about the results in Table 2 is that problems 5 and 6 are more difficult than problems 1 to 4. This is due to the way the weights are generated, being constant for the first two instances, uniformly distributed in instances 3 and 4, and following a Beta distribution in the last two instances. We then observed that because we set $k = 10$ then the T values are on average twice as large as those reported in Table I of Rubinstein (2002). Recall that setting k = 10 was necessary to approach the deviation values reported by Rubinstein (2002).

N	Prob. 1	Prob. 2	Prob. 3	Prob. 4	Prob. 5	Prob. 6
200	3.11	3.26	3.06	3.26	1.06	1.04
400	3.67	4.50	3.50	4.08	2.45	2.56
800	4.65	5.73	4.38	5.12	4.95	5.81
1200	5.88	6.87	5.72	6.32	7.14	8.95
1600	7.79	8.58	7.58	8.30	9.41	11.59
2000	9.63	10.46	9.66	10.16	11.90	14.03

Table 3. Average running times (in seconds) for 20 executions of the CE method

The execution times in Table 3 follow a similar pattern as shown in Table II of Rubinstein (2002). The actual values, however, are significantly different due to the dissimilarity of the computer equipment used here and by Rubinstein (2002). With these data, we are confident that our implementation of the CE method for the max-cut problem—with some changes in the parameter settings—approximates the one suggested in the literature.

In our next experiment we fine-tune the key search parameters for the HCE method. We start by setting K to 100 and then we configure Calibra to use Set 1 for the fine-tuning of c, δ, and α within the following set of values:

$$c = \{0.01, 0.02, 0.03, \ldots, 0.50\}$$
$$\delta = \{0.5, 0.6, 0.7, \ldots, 1.0\}$$
$$\alpha = \{0.5, 0.6, 0.7, \ldots, 1.0\}$$

Calibra suggested the following parameter values for the HCE method: N=0.031n, δ = 0.9, and α = 0.9. Table 4 reports the performance comparison between the CE and HCE methods as adjusted with Calibra. The results in this table are from running both methods (a single time with the same random seed) on the six instances of Set 1. In the table, there is a row for each problem instance in this set. All problem instances have 200 vertices and 19,900 edges. We show the objective function values (Value), the percent deviation from the optimal solution (Dev) and the CPU time in seconds (Time). These values show that both methods are quite capable of tackling the instances in Set 1. The optimal solution to all these artificial problems is 50,000. Regarding number of optimal solutions found, the HCE method performs slightly better (5 out of 6) than the CE method (4 out 6). The average percent deviation slightly favors the CE method. The mechanisms added to the HCE method in order to accelerate the search are evident in the computational times shown in Table 4.

	CE Method			HCE Method		
Instance	Value	Dev (%)	Time	Value	Dev (%)	Time
z1	50000	0.00	5.72	50000	0.00	0.36
z2	50000	0.00	7.31	50000	0.00	0.44
z3	50000	0.00	6.16	50000	0.00	0.38
z4	50000	0.00	7.28	50000	0.00	0.39
z5	49993.1	0.01	6.62	50000	0.00	0.44
z6	49995.8	0.01	7.86	49977.71	0.04	0.45
Average	49998.15	0.00	6.83	49996.29	0.01	0.41

Table 4. Performance comparison between CE and HCE on Set 1

We now apply both methods to the problems in Set 2. This set consists of four problem instances, whose sizes are shown in the second and third column of Table 5. This table also shows the objective function values, the percent deviation from the semidefinite programming upper bound (Goemans and Williamson, 1995) denoted as SDP and the computational time for each of the two methods. The problem instances in Set 2 are significantly larger (in terms of the number of nodes) than those in Set 1 and the graphs in Set 2 are not artificially structured in the same systematic way as in Set 1.

			CE Method			HCE Method			
Instance	|*V*|	|*E*|	Value	Dev	Time	Value	Dev	Time	SDP
g3-15	3375	10125	1.73E+07	94.5%	5218.0	2.69E+08	14.3%	3944.6	3.13E+08
g3-8	512	1536	3.63E+07	20.6%	356.8	4.02E+07	12.2%	8.3	4.57E+07
pm3-15-50	3375	10125	2102	39.5%	5336.2	2895	16.7%	2583. 9	3474
pm3-8-50	512	1536	386	26.8%	271.5	442	16.1%	7.9	527
Average			1.34E+07	45.3%	2795.6	7.72E+07	14.8%	1636.2	

Table 5. Performance comparison between CE and HCE on Set 2

The shortcomings of the CE method are now evident. CE is not able to provide high-quality heuristic solutions to the max-cut problems in Set 2 (as measured by the deviation to a known upper bound). In this test set, HCE is superior both in terms of solution quality and computational time.

So far, we have focused on direct comparisons between the CE and HCE methods. Experiments with Set 1 did not show any discernable performance difference between the two methods. The second experiment provided some evidence that HCE may be more robust than CE when tackling larger, unstructured max-cut problems. Although the intention all along has been to use the max-cut problem as a case for testing a hybridized CE method, it is our obligation to establish—and inform the reader—where the CE and HCE implementations stand in regard to state-of-the-art solution procedures for the max-cut problem. To this end, we consider the benchmarks established by Festa, et al. (2002). Specifically, we compare our implementations to the CirCut heuristic due to Burer et al. (2001), and the GRASP, GRASP with Path Relinking (GPR) and the Variable Neighborhood Search coupled with Path Relinking (VNSPR) due to Festa et al (2002). These methods were run on a SGI Challenge computer at 196 Mhz MIPS R10000 and 7.6 GB of RAM. Van Keken (2005) reports a comparison among different platforms including the aforementioned computer and the 3.2 GHz Intel Xenon that we used in our experiments. His tests show that our computer is approximately 4.52 times faster than the SGI Challenge, so we use this factor in order to normalize the times shown in Tables 7 and 8.

Table 6 summarizes the results obtained from applying the CE and HCE methods to the problem instances in Set 3. The objective function values found by these two methods are shown in the fourth and fifth columns of this table. The remaining columns of objective function values were taken directly from Table 2 in Festa et al. (2002). The last row shows the average percent deviation of the solutions found by each method from the semidefinite programming upper bound (SDP) shown in the last column of the table.

Instance	|*V*|	|*E*|	CE	HCE	CirCut	GRASP	GPR	VNSPR	SDP Bound
G1	800	19176	11419	11584	11624	11540	11563	11621	12078%
G2			11374	11595	11617	11567	11567	11615	12084%
G3			11380	11574	11622	11551	11585	11622	12077%
G11	800	1600	506	552	560	552	564	564	627%
G12			500	542	552	546	552	556	621%
G13			486	564	574	572	580	580	645%
G14	800	4694	2980	3030	3058	3027	3041	3055	3187%
G15			2966	3012	3049	3013	3034	3043	3169%
G16			2959	3015	3045	3013	3028	3043	3172%
G22	2000	19990	10381	13297	13346	13185	13203	13295	14123%
G23			10370	13247	13317	13203	13222	13290	14129%
G24			10381	13250	13314	13165	13242	13276	14131%
G32	2000	4000	176	1336	1390	1370	1392	1396	1560%
G33			146	1314	1360	1348	1362	1376	1537%
G34			128	1342	1368	1348	1364	1372	1541%
G35	2000	11778	6155	7570	7670	7567	7588	7635	8000%
G36			6133	7566	7660	7555	7581	7632	7996%
G37			6154	7585	7666	7576	7602	7643	8009%
G43	1000	9990	6433	6612	6656	6592	6621	6659	7027%
G44			6509	6618	6643	6587	6618	6642	7022%
G45			6427	6592	6652	6598	6620	6646	7020%
G48	3000	6000	3164	6000	6000	6000	6000	6000	6000%
G49			3152	6000	6000	6000	6000	6000	6000%
G50			3178	5880	5880	5862	5880	5880	5988%
Total objective value			123457	149677	150623	149337	149809	150441	157743%
Avg. deviation from bound			21.74	5.11	4.51	5.33	5.03	4.63	

Table 6. Summary of experimental results for instances in Set 3

Table 6 reveals that the quality of the solutions generated by the HCE method is in the same general neighbourhood of those found by the best solution procedures for the max-cut problem that have appeared in the literature. It also reveals that the performance of the original CE method for the max-cut problem is a distant last. The superiority of the CirCut method is evident in Table 6. Only in two cases (G32 and G43) CirCut fails to provide the best-known solution to the problems in Set 3. Moreover, the CirCut method, on average, is the fastest of all (see Table 7). This method is designed to solve binary quadratic programs, in particular the max-cut problem. It is based on a rank-two relaxation scheme that leads to a nonconvex nonlinear optimization problem with number of variables equal to the number of nodes in the graph. A cut is obtained from the solution to the relaxed problem and the solution is perturbed in order to apply the relaxation procedure several times and obtain additional cuts. Only the VNSPR procedure, with numerous search strategies and large computational effort, approaches the quality of the solutions found by CirCut. Regarding computational time, the HCE method is very competitive, as evident in Table 7.

Instance	|*V*|	|*E*|	CE	HCE	CirCut	GRASP	GPR	VNSPR
G1	800	19176	1948.97	43.56	352	467.0	467.0	5029.2
G2			1570.31	40.6	283	450.7	457.3	5026.3
G3			1828.83	41.7	330	454.2	453.1	5285.4
G11	800	1600	1327.38	39.11	74	61.1	63.1	2231.0
G12			1332.52	36.97	58	60.8	62.8	2400.9
G13			1137.44	37.11	62	61.5	63.5	2318.4
G14	800	4694	2294.67	40.57	128	105.8	108.2	3702.2
G15			1437.52	41.64	155	105.8	108.0	3801.8
G16			1437.41	39.41	142	105.8	108.0	3664.2
G22	2000	19990	3714.24	774.46	493	1475.0	1487.6	43728.8
G23			3692.83	680.89	457	1503.3	1493.1	42855.5
G24			3711.16	815.2	521	1495.6	1481.6	43307.3
G32	2000	4000	3719.39	680.67	221	434.1	446.2	18217.9
G33			3651.69	750.34	198	437.8	450.4	16876.5
G34			3692.89	755.22	237	435.2	467.5	17567.7
G35	2000	11778	3808.2	780.65	440	812.2	808.4	36995.8
G36			3944.27	669.51	400	816.8	806.6	36991.8
G37			3850.69	655.45	382	811.9	803.3	37784.5
G43	1000	9990	2835.58	83.21	213	301.1	305.1	7815.0
G44			3633.83	79.72	192	304.6	304.6	7636.9
G45			3641.03	79.45	210	302.9	306.9	7561.7
G48	3000	6000	3729	1566	119	1266.2	1301.1	14317.0
G49			3765.47	1591	134	1363.9	1387.8	14325.0
G50			3783.77	1941	231	1098.9	1127.2	32551.3
Average			2895.38	510.98	251.3	613.84	619.52	17166.34

Table 7. Solution times for problem instances in Set 3

In our final experiment we use the instances in Set 4 to compare the quality of the solutions obtained with the CE, HCE, CirCut, GPR and VNSPR. The graphs in this test set contain positive and negative weight values, a feature exploited by the local search mechanisms in the GRASP and path relinking variants. The objective values obtained in this comparison are shown in Table 8. As before, we executed our computer implementations to obtain the CE and HCE solutions, and we simply reproduce the values reported by Festa, et al. (2002) to populate the rest of the table. The deviation is now computed with respect to the best known solution (Max) since the SDP upper bound is not known for the instances in this set. Unfortunately, Festa et al. (2002) do not report the performance of GRASP on this set of test problems and this is why the corresponding column is missing in Table 8.

For the graphs in Set 4, the HCE method now lags behind the specialized max-cut procedures used for comparison. The HCE method produces solutions of reasonable quality, but it cannot compete with the highly customized methods (which embed a fair amount of problem-specific search mechanisms and strategies). Note that CE does not use any problem-specific knowledge and only the local search is problem-specific in the HCE method. The CE method, as originally proposed for the max-cut problem, cannot tackle the difficult problems in Set 4. CE's average deviation against the best-known solution is just over 70%, rendering the procedure useless for this class of graphs. The relative ranking of the methods according to their average solution time (shown in the last row of Table 8) is the same as in the previous experiment. CirCut is still the fastest method while VNSPR is still the slowest (with an average of more than 6 hours of CPU time).

Instance	$\|V\|$	$\|E\|$	CE	HCE	CirCut	GPR	VNSPR	Max
sg3dl101000	1000	3000	768	860	880	884	892	892
sg3dl102000			782	860	892	896	900	900
sg3dl103000			766	862	882	878	884	884
sg3dl104000			788	860	894	884	896	896
sg3dl105000			762	852	882	868	882	882
sg3dl106000			762	850	886	870	880	886
sg3dl107000			782	854	894	890	896	896
sg3dl108000			748	842	874	876	880	880
sg3dl109000			780	864	890	884	898	898
sg3dl1010000			774	864	886	888	890	890
sg3dl141000	2744	8232	200	2328	2410	2378	2416	2416
sg3dl142000			198	2356	2416	2382	2416	2416
sg3dl143000			184	2342	2408	2390	2406	2408
sg3dl144000			182	2348	2414	2382	2418	2418
sg3dl145000			202	2328	2406	2374	2416	2416
sg3dl146000			194	2338	2412	2390	2420	2420
sg3dl147000			202	2310	2410	2384	2404	2410
sg3dl148000			206	2326	2418	2378	2418	2418
sg3dl149000			192	2312	2388	2362	2384	2388
sg3dl1410000			172	2350	2420	2390	2422	2422
Total objective value			9644	31906	32962	32628	33018	33018
Average deviation			70.81%	3.42%	0.22%	1.24%	0.05%	
Average solution time			3444.6	356.8	236.8	613.8	23496.4	

Table 8. Summary of experimental results for instances in Set 4

5. Conclusions

We embarked on the project of implementing and testing the cross entropy method because (1) the methodology has appeared in several recent publications, (2) it is very intuitive and (3) we were intrigued by the results reported in problems such as the one used here for testing. Our first task was to implement the CE method as described by Rubinstein (2002) in an attempt to reproduce his results. As we describe in the previous section, we were unable to match his reported outcomes exactly. However, after fine-tuning the procedure, our results were similar in terms of solution quality although convergence was achieved with more iterations than those reported by Rubinstein (2002).

Once we gained a good understanding of the CE methodology, we decided to develop an extension that includes hybridization with local search and systematic diversification. Our motivation was to develop a general way of incorporating problem context information and diversification strategies in order to make the hybridized version competitive with specialized procedures. The effect of these changes was to accelerate convergence and reduce sample sizes. Our ideas are put together as a general way of creating a hybrid procedure that is based on the CE methodology. We used the max-cut problem to illustrate the process of developing the hybrid method that we labeled HCE. Our experiments show that the HCE implementation is competitive with a specialized GRASP for the same problem. It is not competitive, however, with more sophisticated procedures specifically tailored for the max-cut problem (i.e., GRASP and VNS hybrids and Circut). It should be pointed out that this level of sophistication was achieved at a high cost in terms of design and implementation time. If this total time is important, then HCE becomes a very attractive alternative.

Acknowledgments

This research has been partially supported by the *Ministerio de Educación y Ciencia* of Spain (Grant Refs. TIN2006-02696 TIN2005-08943-C02-02), Generalitat Valenciana (GV/2007/047) and by the Comunidad de Madrid – Universidad Rey Juan Carlos project (Ref. URJC-CM-2006-CET-0603). We would like to thank Mr. Camilo Sesto for sharing his wisdom and making this project particularly enjoyable.

References

Adenso-Díaz, B. and M. Laguna (2006) "Fine-tuning of Algorithms Using Partial Experimental Designs and Local Search," *Operations Research*, vol. 54, no. 1, pp. 99-114.

Alon, G., D. P. Kroese, T. Raviv and R. Y. Rubinstein (2005) "Application of the Cross-Entropy Method to the Buffer Allocation Problem in a Simulation-Based Environment," *Annals of Operations Research*, vol. 134, pp. 137-151.

Burer, S., R. D. C. Monteiro and Y. Zang (2001) "Rank-two Relaxation Heuristics for Max-cut and Other Binary Quadratic Programs," *SIAM Journal on Optimization*, vol. 12, pp. 503-521.

Chepuri, K. and T. Homem-de-Mello (2005) "Solving the Vehicle Routing Problem with Stochastic Demands using the Cross-Entropy Method," *Annals of Operations Research*, vol. 134, pp. 153-181.

de Boer, P-T, D. P. Kroese, S. Mannor and R. Y. Rubinstein (2005a) "The Cross-Entropy Method for Combinatorial Optimization, Rare Event Simulation and Neural Computation," *Annals of Operations Research*, vol. 134.

de Boer, P-T, D. P. Kroese, S. Mannor and R. Y. Rubinstein (2005b) "A Tutorial of the Cross Entropy Method," *Annals of Operations Research*, vol. 134, pp. 19-67.

Festa, P., P. M. Pardalos, M. G. C. Resende and C. C. Ribeiro (2002) "Randomized Heuristics for the Max-cut Problem," *Optimization Methods and Software*, vol. 7, pp. 1033-1058.

Goemans, M. X. and D. P. Williams (1995) "Improved Approximation Algorithms for the Max-cut and Satisfiability Problems using Semidefinite Programming," *Journal of the ACM*, vol. 42, pp. 1115-1145.

Helmberg, C. and F. Rendl (2000) "A Spectral Bundle Method for Semidefinite Programming," *SIAM Journal on Optimization*, vol. 10, pp. 673-696.

Rubinstein, R. Y. (1997) "Optimization of Computer Simulation Models with Rare Events," *European Journal of Operations Research*, vol. 99, pp. 89–112.

Rubinstein, R. Y. (1999) "The Simulated Entropy Method for Combinatorial and Continuous Optimization," *Methodology and Computing in Applied Probability*, vol. 2, pp. 127–190.

Rubinstein, R. Y. (2001) “Combinatorial Optimization, Cross-Entropy, Ants and Rare Events,” in *Stochastic Optimization: Algorithms and Applications*, S. Uryasev and P. M. Pardalos (eds.), Kluwer Academic Publishers, pp. 304–358.

Rubinstein, R.Y. (2002) “Cross-Entropy and Rare Events for Maximal Cut and Partition Problems,” *ACM Transaction on Modeling and Computer Simulation*, vol. 12, no. 1, pp. 27-53.

Van Keken, P. (2005) “Linux PC Single CPU Benchmark Comparison” http://www.geo.lsa.umich.edu/~keken/benchmarks/single_cpu.html