Annals of Operations Research 134, 19–67, 2005
© 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

A Tutorial on the Cross-Entropy Method

PIETER-TJERK DE BOER * ptdeboer@cs.utwente.nl
Department of Electrical Engineering, Mathematics and Computer Science, University of Twente, Enschede, The Netherlands

DIRK P. KROESE kroese@maths.uq.edu.au
Department of Mathematics, The University of Queensland, Brisbane 4072, Australia

SHIE MANNOR shie@ece.mcgill.ca
Department of Electrical and Computer Engineering, McGill University, 3480 University Street, Montreal, Quebec, Canada

REUVEN Y. RUBINSTEIN ierrr01@ie.technion.ac.il
Department of Industrial Engineering, Technion, Israel Institute of Technology, Haifa 32000, Israel

Received January 2003; Revised November 2003; Accepted January 2004

Abstract. The cross-entropy (CE) method is a new generic approach to combinatorial and multi-extremal optimization and rare event simulation. The purpose of this tutorial is to give a gentle introduction to the CE method. We present the CE methodology, the basic algorithm and its modifications, and discuss applications in combinatorial optimization and machine learning.

Keywords: cross-entropy method, Monte-Carlo simulation, randomized optimization, machine learning, rare events

Many everyday tasks in operations research involve solving complicated optimization problems. The travelling salesman problem (TSP), the quadratic assignment problem (QAP) and the max-cut problem are a representative sample of combinatorial optimization problems (COP) where the problem being studied is completely known and static. In contrast, the buffer allocation problem (BAP) is a *noisy* estimation problem where the objective function needs to be estimated since it is unknown. Discrete event simulation is one method for estimating an unknown objective function.

The purpose of this tutorial is to show that the CE method provides a simple, efficient and general method for solving such problems. Moreover, we wish to show that the CE method is also valuable for *rare event simulation*, where very small probabilities need to be accurately estimated – for example in reliability analysis, or performance analysis of telecommunication systems. This tutorial is intended for a broad audience of operations research specialists, computer scientists, mathematicians, statisticians and engineers. Our aim is to explain the foundations of the CE method and consider various applications.

*Corresponding author.

GOOGLE EXHIBIT 1007
Google v. K.Mizra
IPR2026-00304

The CE method was motivated by an adaptive algorithm for estimating probabilities of rare events in complex stochastic networks (Rubinstein, 1997), which involves variance minimization. It was soon realized (Rubinstein, 1999, 2001) that a simple cross-entropy modification of Rubinstein (1997) could be used not only for estimating probabilities of rare events but for solving difficult COPs as well. This is done by translating the "deterministic" optimization problem into a related "stochastic" optimization problem and then using rare event simulation techniques similar to Rubinstein (1997). Several recent applications demonstrate the power of the CE method (Rubinstein, 1999) as a generic and practical tool for solving NP-hard problems.

The CE method involves an iterative procedure where each iteration can be broken down into two phases:

1. Generate a random data sample (trajectories, vectors, etc.) according to a specified mechanism.
2. Update the parameters of the random mechanism based on the data to produce a "better" sample in the next iteration.

The significance of the CE method is that it defines a precise mathematical framework for deriving fast, and in some sense "optimal" updating/learning rules, based on advanced simulation theory. Other well-known randomized methods for combinatorial optimization problems are simulated annealing (Aarts and Korst, 1989), tabu search (Glover and Laguna, 1993), and genetic algorithms (Goldberg, 1989). Recent related work on randomized combinatorial optimization includes the nested partitioning method (Shi and Olafsson, 2000) and the ant colony optimization meta-heuristic (Colorni et al., 1996; Dorigo, Di Caro, and Gambardella, 1999; Gutjahr, 2000).

Many COPs can be formulated as optimization problems concerning a weighted graph. As mentioned before, in CE a deterministic optimization problem is translated into an associated *stochastic* optimization problem. Depending on the particular problem, we introduce randomness in either (a) the nodes or (b) the edges of the graph. We speak of *stochastic node networks* (SNN) in the former case and *stochastic edge networks* (SEN) in the latter. Examples of SNN problems are the maximal cut (max-cut) problem, the buffer allocation problem and clustering problems. Examples of SEN problems are the travelling salesman problem, the quadratic assignment problem, the clique problem, and optimal policy search in Markovian decision problems (MDPs).

It should be emphasized that the CE method may be successfully applied to both deterministic and stochastic COPs. In the latter the objective function itself is random or needs to be estimated via simulation. Stochastic COPs typically occur in stochastic scheduling, flow control and routing of data networks and in various simulation-based optimization problems (Rubinstein and Melamed, 1998), such as the optimal buffer allocation problem (Alon et al., 2005).

Estimation of the probability of rare events is essential for guaranteeing adequate performance of engineering systems. For example, consider a telecommunications system that accepts calls from many customers. Under normal operating conditions each

client may be rejected with a very small probability. Naively, in order to estimate this small probability we would need to simulate the system under normal operating conditions for a long time. A better way to estimate this probability is to use *importance sampling* (IS), which is a well-known variance reduction technique in which the system is simulated under a different set of parameters – or, more generally, a different probability distribution – so as to make the occurrence of the rare event more likely. A major drawback of the IS technique is that the optimal reference (also called tilting) parameters to be used in IS are usually very difficult to obtain. The advantage of the CE method is that it provides a simple adaptive procedure for estimating the optimal reference parameters. Moreover, the CE method also enjoys asymptotic convergence properties. For example, it is shown in Homem-de-Mello and Rubinstein (2002) that for *static* models – cf. Remark 2.1 – under mild regularity conditions the CE method terminates with probability 1 in a finite number of iterations, and delivers a consistent and asymptotically normal estimator for the optimal reference parameters. Recently the CE method has been successfully applied to the estimation of rare event probabilities in dynamic models, in particular queueing models involving both *light* and *heavy* tail input distributions (de Boer, Kroese and Rubinstein, 2002; Kroese and Rubinstein, 2004). In addition to rare event simulation and combinatorial optimization, the CE method can be efficiently applied for continuous multi-extremal optimization, see Rubinstein (1999) and the recent CE book (Rubinstein and Kroese, 2004).

An increasing number of applications is being found for the CE method. Recent publications on applications of the CE method include: buffer allocation (Alon et al., 2005); static simulation models (Homem-de-Mello and Rubinstein, 2002); queueing models of telecommunication systems (de Boer, 2000; de Boer, Kroese, and Rubinstein, 2004); neural computation (Dubin, 2002, 2004); control and navigation (Helvik and Wittner, 2001); DNA sequence alignment (Keith and Kroese, 2002); scheduling (Margolin, 2002, 2004); vehicle routing (Chepuri and Homem-de-Mello, 2005); reinforcement learning (Mannor, Rubinstein, and Gat, 2003; Menache, Mannor, and Shimkin, 2005); project management (Cohen, Golany, and Shtub, 2005); heavy-tail distributions (Asmussen, Kroese, and Rubinstein, 2005); (Kroese and Rubinstein, 2004); CE convergence (Margolin, 2005); network reliability (Hui et al., 2005); repairable systems (Ridder, 2005); and max-cut and bipartition problems (Rubinstein, 2002).

It is not our intention here to compare the CE method with other heuristics. Our intention is mainly to demonstrate its beauty and simplicity and promote CE for further applications to combinatorial and multi-extremal optimization and rare event simulation.

The rest of the tutorial is organized as follows. In Section 1 we present two toy examples that illustrate the basic methodology behind the CE method. The general theory and algorithms are detailed in Section 2. In Section 3 we discuss various applications and examples of using the CE method for solving COPs. In Section 4 two useful modifications of the CE method are discussed. Further developments are briefly reviewed in Section 5.

The CE *home page*, featuring many links, articles, references, tutorials and computer programs on CE, can be found at: `http://www.cemethod.org/`

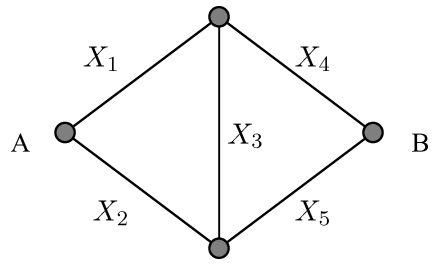

Figure 1. Shortest path from A to B.

1. Methodology: Two examples

In this section we illustrate the methodology of the CE method via two toy examples; one dealing with rare event simulation, and the other with combinatorial optimization.

1.1. A rare event simulation example

Consider the weighted graph of figure 1, with random weights $X_1, \ldots, X_5$. Suppose the weights are independent of each other and are exponentially distributed with means $u_1, \ldots, u_5$, respectively. Define $\boldsymbol{X} = (X_1, \ldots, X_5)$ and $\boldsymbol{u} = (u_1, \ldots, u_5)$. Denote the probability density function (pdf) of $\boldsymbol{X}$ by $f(\cdot; \boldsymbol{u})$; thus,

$$f(\boldsymbol{x}; \boldsymbol{u}) = \exp\left(-\sum_{j=1}^{5} \frac{x_j}{u_j}\right) \prod_{j=1}^{5} \frac{1}{u_j}. \tag{1}$$

Let $S(\boldsymbol{X})$ be the total length of the shortest path from node A to node B. We wish to estimate from simulation

$$\ell = \mathbb{P}(S(\boldsymbol{X}) \geq \gamma) = \mathbb{E} I_{\{S(\boldsymbol{X}) \geq \gamma\}}, \tag{2}$$

that is, the probability that the length of the shortest path $S(\boldsymbol{X})$ will exceed some fixed γ.

A straightforward way to estimate ℓ in (2) is to use *Crude Monte Carlo* (CMC) simulation. That is, we draw a random sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ from the distribution of $\boldsymbol{X}$ and use

$$\frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \gamma\}} \tag{3}$$

as the unbiased estimator of ℓ. However, for large γ the probability ℓ will be very small and CMC requires a very large simulation effort, that is, N needs to be very large in order to estimate ℓ accurately – that is, to obtain a small relative error, for example of 1%. A better way is to perform the simulation is to use *importance sampling* (IS). That is, let

g be another probability density such that $g(\boldsymbol{x}) = 0 \;\Rightarrow\; I_{\{S(\boldsymbol{x})\geq\gamma\}} f(\boldsymbol{x}) = 0$. Using the density g we can represent ℓ as

$$\ell = \int I_{\{S(\boldsymbol{x})\geq\gamma\}} \frac{f(\boldsymbol{x})}{g(\boldsymbol{x})}\, g(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} = \mathbb{E}_g I_{\{S(\boldsymbol{X})\geq\gamma\}} \frac{f(\boldsymbol{X})}{g(\boldsymbol{X})}, \tag{4}$$

where the subscript g means that the expectation is taken with respect to g, which is called the *importance sampling* (IS) density. An unbiased estimator of ℓ is

$$\hat{\ell} = \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i)\geq\gamma\}}\, W(\boldsymbol{X}_i), \tag{5}$$

where $\hat{\ell}$ is called the *importance sampling* (IS) or the *likelihood ratio* (LR) estimator,

$$W(\boldsymbol{x}) = f(\boldsymbol{x})/g(\boldsymbol{x}) \tag{6}$$

is called the *likelihood ratio* (LR), and $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ is a *random sample* from g, that is, $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ are iid random vectors with density g. In the particular case where there is no "change of measure", i.e., $g = f$, we have $W = 1$, and the LR estimator in (6) reduces to the CMC estimator (3).

If we restrict ourselves to g such that $X_1, \ldots, X_5$ are independent and exponentially distributed with means $v_1, \ldots, v_5$, then

$$W(\boldsymbol{x}; \boldsymbol{u}, \mathbf{v}) := \frac{f(\boldsymbol{x}; \boldsymbol{u})}{f(\boldsymbol{x}; \mathbf{v})} = \exp\left(-\sum_{j=1}^{5} x_j \left(\frac{1}{u_j} - \frac{1}{v_j}\right)\right) \prod_{j=1}^{5} \frac{v_j}{u_j}. \tag{7}$$

In this case the "change of measure" is determined by the parameter vector $\mathbf{v} = (v_1, \ldots, v_5)$. The main problem now is how to select a $\mathbf{v}$ which gives the most accurate estimate of ℓ for a given simulation effort. One of the strengths of the CE method for rare event simulation is that it provides a fast way to determine/estimate the optimal parameters. To this end, without going into the details, a quite general CE algorithm for rare event estimation is outlined below.

Algorithm 1.1

1. Define $\hat{\mathbf{v}}_0 := \boldsymbol{u}$. Set $t := 1$ (iteration counter).
2. Generate a random sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ according to the pdf $f(\cdot; \hat{\mathbf{v}}_{t-1})$. Calculate the performances $S(\boldsymbol{X}_i)$ for all i, and order them from smallest to biggest, $S_{(1)} \leq \cdots \leq S_{(N)}$. Let $\hat{\gamma}_t$ be the $(1-\rho)$ sample quantile of performances: $\hat{\gamma}_t := S_{(\lceil (1-\rho)N \rceil)}$, provided this is less than γ. Otherwise, put $\hat{\gamma}_t := \gamma$.
3. Use the **same** sample to calculate, for $j = 1, \ldots, n\ (= 5)$,

$$\hat{v}_{t,j} = \frac{\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i)\geq\hat{\gamma}_t\}} W(\boldsymbol{X}_i; \boldsymbol{u}, \hat{\mathbf{v}}_{t-1})\, X_{ij}}{\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i)\geq\hat{\gamma}_t\}} W(\boldsymbol{X}_i; \boldsymbol{u}, \hat{\mathbf{v}}_{t-1})}.$$

Table 1
Convergence of the sequence $\{(\hat{\gamma}_t, \hat{\mathbf{v}}_t)\}$.

t	$\hat{\gamma}_t$	$\hat{\mathbf{v}}_t$				
0		0.250	0.400	0.100	0.300	0.200
1	0.575	0.513	0.718	0.122	0.474	0.335
2	1.032	0.873	1.057	0.120	0.550	0.436
3	1.502	1.221	1.419	0.121	0.707	0.533
4	1.917	1.681	1.803	0.132	0.638	0.523
5	2.000	1.692	1.901	0.129	0.712	0.564

4. If $\hat{\gamma}_t = \gamma$ then proceed to step 5; otherwise set $t := t + 1$ and reiterate from step 2.
5. Let T be the final iteration. Generate a sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{N_1}$ according to the pdf $f(\cdot; \hat{\mathbf{v}}_T)$ and estimate ℓ via the IS estimate

$$\hat{\ell} = \frac{1}{N_1} \sum_{i=1}^{N_1} I_{\{S(\boldsymbol{X}_i) \geq \gamma\}} W(\boldsymbol{X}_i; \boldsymbol{u}, \hat{\mathbf{v}}_T).$$

Note that in steps 2–4 the optimal IS parameter is estimated. In the final step (step 5) this parameter is used to estimate the probability of interest. We need to supply the fraction ρ (typically between 0.01 and 0.1) and the parameters N and N_1 in advance.

As an example, consider the case where the *nominal* parameter vector $\boldsymbol{u}$ is given by (0.25, 0.4, 0.1, 0.3, 0.2). Suppose we wish to estimate the probability that the minimum path is greater than $\gamma = 2$. Crude Monte Carlo with 10^7 samples gave an estimate $1.65 \cdot 10^{-5}$ with an estimated *relative error*, RE, (that is, $\mathrm{Var}(\hat{\ell})^{1/2}/\ell$) of 0.165. With 10^8 samples we got the estimate $1.30 \cdot 10^{-5}$ with RE 0.03.

Table 1 displays the results of the CE method, using $N = 1{,}000$ and $\rho = 0.1$. This table was computed in less than half a second.

Using the estimated optimal parameter vector of $\hat{\mathbf{v}}_5 = (1.692, 1.901, 0.129, 0.712, 0.564)$, the final step with $N_1 = 10^5$ gave now an estimate of $1.34 \cdot 10^{-5}$ with an estimated RE of 0.03. The simulation time was only 3 seconds, using a Matlab implementation on a Pentium III 500 MHz processor. In contrast, the CPU time required for the CMC method with 10^7 samples is approximately 630 seconds, and with 10^8 samples approximately 6350. We see that with a minimal amount of work we have reduced our simulation effort (CPU time) by roughly a factor of 2000.

1.2. A combinatorial optimization example

Consider a binary vector $\boldsymbol{y} = (y_1, \ldots, y_n)$. Suppose that we do not know which components of $\boldsymbol{y}$ are 0 and which are 1. However, we have an "oracle" which for each binary

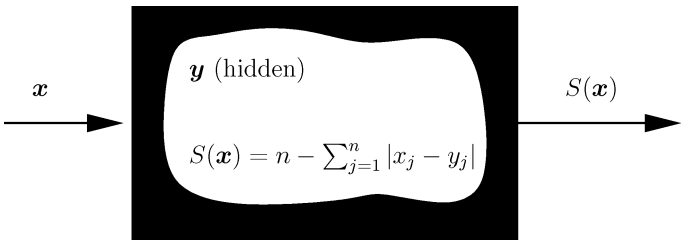

Figure 2. A black box for decoding vector $\boldsymbol{y}$.

input vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ returns the performance or response,

$$S(\boldsymbol{x}) = n - \sum_{j=1}^{n} |x_j - y_j|,$$

representing the number of *matches* between the elements of $\boldsymbol{x}$ and $\boldsymbol{y}$. Our goal is to present a random search algorithm which reconstructs[1] (decodes) the unknown vector $\boldsymbol{y}$ by maximizing the function $S(\boldsymbol{x})$ on the space of n-dimensional binary vectors.

A naive way to find $\boldsymbol{y}$ is to repeatedly generate binary vectors $\boldsymbol{X} = (X_1, \ldots, X_n)$ such that $X_1, \ldots, X_n$ are independent Bernoulli random variables with success probabilities $p_1, \ldots, p_n$. We write $\boldsymbol{X} \sim \mathsf{Ber}(\boldsymbol{p})$, where $\boldsymbol{p} = (p_1, \ldots, p_n)$. Note that if $\boldsymbol{p} = \boldsymbol{y}$, which corresponds to the degenerate case of the Bernoulli distribution, we have $S(\boldsymbol{X}) = n$, $\boldsymbol{X} = \boldsymbol{y}$, and the naive search algorithm yields the optimal solution with probability 1. The CE method for combinatorial optimization consists of casting the underlying problem into the rare event framework (2) and then creating a sequence of parameter vectors $\hat{\boldsymbol{p}}_0, \hat{\boldsymbol{p}}_1, \ldots$ and *levels* $\hat{\gamma}_1, \hat{\gamma}_2, \ldots$, such that the sequence $\hat{\gamma}_1, \hat{\gamma}_2, \ldots$ converges to the optimal performance (n here) and the sequence $\hat{\boldsymbol{p}}_0, \hat{\boldsymbol{p}}_1, \ldots$ converges to the optimal parameter vector ($\boldsymbol{y}$ here). Again, the CE procedure – which is similar to the rare event procedure described in Algorithm 1.1 – is outlined below, without detail.

Algorithm 1.2

1. Start with some $\hat{\boldsymbol{p}}_0$, say $\hat{\boldsymbol{p}}_0 = (1/2, 1/2, \ldots, 1/2)$. Let $t := 1$.
2. Draw a sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ of Bernoulli vectors with success probability vector $\hat{\boldsymbol{p}}_{t-1}$. Calculate the performances $S(\boldsymbol{X}_i)$ for all i, and order them from smallest to biggest, $S_{(1)} \leq \cdots \leq S_{(N)}$. Let $\hat{\gamma}_t$ be $(1 - \rho)$ sample quantile of the performances: $\hat{\gamma}_t = S_{(\lceil (1-\rho)N \rceil)}$.

[1] Of course, in this toy example the vector $\boldsymbol{y}$ can be easily reconstructed from the input vectors $(0, 0, \ldots, 0)$, $(1, 0, \ldots, 0)$, $(0, 1, 0, \ldots, 0), \ldots, (0, \ldots, 0, 1)$ only.

Table 2
The convergence of the parameter vector.

t	$\hat{\gamma}_t$	$\hat{\boldsymbol{p}}_t$									
0		0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50
1	7	0.60	0.40	0.80	0.40	1.00	0.00	0.20	0.40	0.00	0.00
2	9	0.80	0.80	1.00	0.80	1.00	0.00	0.00	0.40	0.00	0.00
3	10	1.00	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00
4	10	1.00	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00

3. Use the **same** sample to calculate $\hat{\boldsymbol{p}}_t = (\hat{p}_{t,1}, \ldots, \hat{p}_{t,n})$ via

$$\hat{p}_{t,j} = \frac{\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \hat{\gamma}_t\}} \, I_{\{X_{ij}=1\}}}{\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \hat{\gamma}_t\}}}, \tag{8}$$

   $j = 1, \ldots, n$, where $\boldsymbol{X}_i = (X_{i1}, \ldots, X_{in})$.

4. If the **stopping criterion** is met, then **stop**; otherwise set $t := t + 1$ and reiterate from step 2.

A possible stopping criterion is to stop when $\hat{\gamma}_t$ does not change for a number of subsequent iterations. Another possible stopping criterion is to stop when the vector $\hat{\boldsymbol{p}}_t$ has converged to a degenerate – that is, binary – vector.

Note that the interpretation of 8 is very simple: to *update* the jth success probability we count how many vectors of the last sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ have a performance greater than or equal to $\hat{\gamma}_t$ *and* have the jth coordinate equal to 1, and we divide (normalize) this by the number of vectors that have a performance greater than or equal to $\hat{\gamma}_t$.

As an example, consider the case $n = 10$, where $\boldsymbol{y} = (1, 1, 1, 1, 1, 0, 0, 0, 0, 0)$. Using a sample $N = 50$, $\rho = 0.1$ and the initial parameter vector $\hat{\boldsymbol{p}}_0 = (1/2, 1/2, \ldots, 1/2)$, Algorithm 1.2 yields the results given in Table 2.

We see that the $\hat{\boldsymbol{p}}_t$ and $\hat{\gamma}_t$ converge very quickly to the optimal degenerated CE parameter vector $\boldsymbol{p}^* = \boldsymbol{y}$ and optimal performance $\gamma^* = n$, respectively.

Remark 1.1 (Likelihood ratio term). Note that algorithms 1.1 and 1.2 are almost the *same*. The most important difference is the absence of the likelihood ratio term W in step 3 of Algorithm 1.2. The reason is that the choice of the initial parameter vector $\hat{\boldsymbol{p}}_0$ is *quite arbitrary*, so using W would be meaningless, while in rare event simulation it is an essential part of the estimation problem. For more details see Remark 2.4 below.

2. The main algorithm(s)

2.1. The CE method for rare event simulation

In this subsection we discuss the main ideas behind the CE algorithm for rare event simulation. When reading this section, the reader is encouraged to refer back to the toy example presented in Section 1.1.

Let $\boldsymbol{X} = (X_1, \ldots, X_n)$ be a random vector taking values in some space $\mathcal{X}$. Let $\{f(\cdot; \mathbf{v})\}$ be a family of *probability density functions* (pdfs) on $\mathcal{X}$, with respect to some base measure ν. Here $\mathbf{v}$ is a real-valued parameter (vector). Thus,

$$\mathbb{E}H(\boldsymbol{X}) = \int_{\mathcal{X}} H(\boldsymbol{x})\, f(\boldsymbol{x}; \mathbf{v})\, \nu(d\boldsymbol{x}),$$

for any (measurable) function H. In most (or all) applications ν is either a counting measure or the Lebesgue measure. In the former case f is often called a probability mass function, but in this paper we will always use the generic terms density or pdf. For the rest of this section we take for simplicity $\mathbf{v}(d\boldsymbol{x}) = d\boldsymbol{x}$.

Let S be some real-valued function on $\mathcal{X}$. Suppose we are interested in the probability that $S(\boldsymbol{x})$ is greater than or equal to some real number γ, under $f(\cdot; \boldsymbol{u})$. This probability can be expressed as

$$\ell = \mathbb{P}_{\boldsymbol{u}}(S(\boldsymbol{X}) \geq \gamma) = \mathbb{E}_{\boldsymbol{u}}\, I_{\{S(\boldsymbol{X}) \geq \gamma\}}.$$

If this probability is very small, say smaller than 10^{-5}, we call $\{S(\boldsymbol{X}) \geq \gamma\}$ a *rare event*.

A straightforward way to estimate ℓ is to use crude Monte-Carlo simulation: Draw a random sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ from $f(\cdot; \boldsymbol{u})$; then

$$\frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \gamma\}}$$

is an unbiased estimator of ℓ. However this poses serious problems when $\{S(\boldsymbol{X}) \geq \gamma\}$ is a rare event. In that case a large simulation effort is required in order to estimate ℓ accurately, i.e., with small relative error or a narrow confidence interval.

An alternative is based on importance sampling: take a random sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ from an *importance sampling* (different) density g on $\mathcal{X}$, and evaluate ℓ using the LR estimator (see (5))

$$\hat{\ell} = \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \gamma\}} \frac{f(\boldsymbol{X}_i; \boldsymbol{u})}{g(\boldsymbol{X}_i)}. \tag{9}$$

It is well known that the best way to estimate ℓ is to use the change of measure with density

$$g^*(\boldsymbol{x}) := \frac{I_{\{S(\boldsymbol{x}) \geq \gamma\}} f(\boldsymbol{x}; \boldsymbol{u})}{\ell}. \tag{10}$$

Namely, by using this change of measure we have in (9)

$$I_{\{S(\boldsymbol{X}_i)\geq\gamma\}}\,\frac{f(\boldsymbol{X}_i;\boldsymbol{u})}{g^*(\boldsymbol{X}_i)}=\ell,$$

for all i. In other words, the estimator (9) has zero variance, and we need to produce only $N=1$ sample.

The obvious difficulty is of course that this g^* depends on the unknown parameter ℓ. Moreover, it is often convenient to choose a g in the family of densities $\{f(\cdot;\mathbf{v})\}$. The idea now is to choose the parameter vector, called the *reference parameter* (sometimes called tilting parameter) $\mathbf{v}$ such that the *distance* between the densities g^* and $f(\cdot;\mathbf{v})$ is minimal. A particular convenient measure of distance between two densities g and h is the *Kullback-Leibler distance*, which is also termed the *cross-entropy* between g and h. The Kullback-Leibler distance is defined as:

$$\mathcal{D}(g,h)=\mathbb{E}_g\ln\frac{g(\boldsymbol{X})}{h(\boldsymbol{X})}\;=\int g(\boldsymbol{x})\ln g(\boldsymbol{x})\,d\boldsymbol{x}-\int g(\boldsymbol{x})\ln h(\boldsymbol{x})\,d\boldsymbol{x}.$$

We note that $\mathcal{D}$ is not a "distance" in the formal sense; for example, it is not symmetric.

Minimizing the Kullback-Leibler distance between g^* in (10) and $f(\cdot;\mathbf{v})$ is equivalent to choosing $\mathbf{v}$ such that $-\int g^*(\boldsymbol{x})\ln f(\boldsymbol{x};\mathbf{v})\,d\boldsymbol{x}$ is minimized, which is equivalent to solving the maximization problem

$$\max_{\mathbf{v}}\int g^*(\boldsymbol{x})\ln f(\boldsymbol{x};\mathbf{v})\,d\boldsymbol{x}. \tag{11}$$

Substituting g^* from (10) into (11) we obtain the maximization program

$$\max_{\mathbf{v}}\int\frac{I_{\{S(\boldsymbol{x})\geq\gamma\}}f(\boldsymbol{x};\boldsymbol{u})}{\ell}\ln f(\boldsymbol{x};\mathbf{v})\,d\boldsymbol{x}, \tag{12}$$

which is equivalent to the program

$$\max_{\mathbf{v}}D(\mathbf{v})=\max_{\mathbf{v}}\mathbb{E}_{\boldsymbol{u}}\,I_{\{S(\boldsymbol{X})\geq\gamma\}}\ln f(\boldsymbol{X};\mathbf{v}), \tag{13}$$

where D is implicitly defined above. Using again importance sampling, with a change of measure $f(\cdot;\boldsymbol{w})$ we can rewrite (13) as

$$\max_{\mathbf{v}}D(\mathbf{v})=\max_{\mathbf{v}}\;\mathbb{E}_{\boldsymbol{w}}\,I_{\{S(\boldsymbol{X})\geq\gamma\}}\,W(\boldsymbol{X};\boldsymbol{u},\boldsymbol{w})\ln f(\boldsymbol{X};\mathbf{v}), \tag{14}$$

for *any* reference parameter $\boldsymbol{w}$, where

$$W(\boldsymbol{x};\boldsymbol{u},\boldsymbol{w})=\frac{f(\boldsymbol{x};\boldsymbol{u})}{f(\boldsymbol{x};\boldsymbol{w})}$$

is the *likelihood ratio*, at $\boldsymbol{x}$, between $f(\cdot;\boldsymbol{u})$ and $f(\cdot;\boldsymbol{w})$. The optimal solution of (14) can be written as

$$\mathbf{v}^* = \underset{\mathbf{v}}{\operatorname{argmax}}\ \mathbb{E}_{\boldsymbol{w}}\, I_{\{S(\boldsymbol{X})\geq\gamma\}}\, W(\boldsymbol{X};\boldsymbol{u},\boldsymbol{w}) \ln f(\boldsymbol{X};\mathbf{v}). \tag{15}$$

We may *estimate* $\mathbf{v}^*$ by solving the following stochastic program (also called *stochastic counterpart* of (14))

$$\max_{\mathbf{v}} \hat{D}(\mathbf{v}) = \max_{\mathbf{v}} \frac{1}{N}\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i)\geq\gamma\}}\, W(\boldsymbol{X}_i;\boldsymbol{u},\boldsymbol{w})\, \ln f(\boldsymbol{X}_i;\mathbf{v}), \tag{16}$$

where $\boldsymbol{X}_1,\ldots,\boldsymbol{X}_N$ is a random sample from $f(\cdot;\boldsymbol{w})$. In typical applications the function $\hat{D}$ in (16) is convex and differentiable with respect to $\mathbf{v}$, see also Rubinstein and Shapiro (1993) and, thus, the solution of (16) may be readily obtained by solving (with respect to $\mathbf{v}$) the following system of equations:

$$\frac{1}{N}\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i)\geq\gamma\}}\, W(\boldsymbol{X}_i;\boldsymbol{u},\boldsymbol{w})\, \nabla \ln f(\boldsymbol{X}_i;\mathbf{v}) = \mathbf{0}, \tag{17}$$

where the gradient is with respect to $\mathbf{v}$.

The advantage of this approach is that the solution of (17) can often be calculated *analytically*. In particular, this happens if the distributions of the random variables belong to a *natural exponential family* (NEF). For further details see Rubinstein and Kroese (2004) and Example 2.1 below.

It is important to note that with $\boldsymbol{w} = \boldsymbol{u}$ the CE program (16) is useful only in the case where the probability of the "target event" $\{S(\boldsymbol{X}) \geq \gamma\}$ is not too small, say $\ell \geq 10^{-5}$. For rare event probabilities, however (when, say, $\ell < 10^{-5}$), the program (16) is difficult to carry out. Namely, due to the rareness of the events $\{S(\boldsymbol{X}_i) \geq \gamma\}$, most of the indicator random variables $I_{\{S(\boldsymbol{X}_i)\geq\gamma\}},\ i = 1,\ldots,N$ will be zero, for moderate N. The same holds for the derivatives of $\hat{D}(\mathbf{v})$ as given in the left-hand side of (17).

A *multi-level* algorithm can be used to overcome this difficulty. The idea is to construct a sequence of reference parameters $\{\mathbf{v}_t,\ t \geq 0\}$ and a sequence of levels $\{\gamma_t,\ t \geq 1\}$, and iterate in both γ_t and $\mathbf{v}_t$ (see Algorithm 2.1 below).

We initialize by choosing a not very small ρ, say $\rho = 10^{-2}$ and by defining $\mathbf{v}_0 = \boldsymbol{u}$. Next, we let γ_1 ($\gamma_1 < \gamma$) be such that, under the original density $f(\boldsymbol{x};\boldsymbol{u})$, the probability $\ell_1 = \mathbb{E}_{\boldsymbol{u}} I_{\{S(\boldsymbol{X})\geq\gamma_1\}}$ is at least ρ. We then let $\mathbf{v}_1$ be the optimal CE reference parameter for estimating ℓ_1, and repeat the last two steps iteratively with the goal of estimating the pair $\{\ell, \mathbf{v}^*\}$. In other words, each iteration of the algorithm consists of two main *phases*. In the first phase γ_t is updated, in the second $\mathbf{v}_t$ is updated. Specifically, starting with $\mathbf{v}_0 = \boldsymbol{u}$ we obtain the subsequent γ_t and $\mathbf{v}_t$ as follows:

1. **Adaptive updating of** γ_t. For a fixed $\mathbf{v}_{t-1}$, let γ_t be a $(1-\rho)$-*quantile* of $S(\boldsymbol{X})$ under $\mathbf{v}_{t-1}$. That is, γ_t satisfies

$$\mathbb{P}_{\mathbf{v}_{t-1}}(S(\boldsymbol{X}) \geq \gamma_t) \geq \rho, \tag{18}$$

$$\mathbb{P}_{\mathbf{v}_{t-1}}(S(\boldsymbol{X}) \leq \gamma_t) \geq 1-\rho, \tag{19}$$

where $\boldsymbol{X} \sim f(\cdot; \mathbf{v}_{t-1})$.
A simple estimator $\hat{\gamma}_t$ of γ_t can be obtained by drawing a random sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ from $f(\cdot; \mathbf{v}_{t-1})$, calculating the performances $S(\boldsymbol{X}_i)$ for all i, ordering them from smallest to biggest: $S_{(1)} \leq \cdots \leq S_{(N)}$ and finally, evaluating the $(1-\rho)$ sample quantile as

$$\hat{\gamma}_t = S_{(\lceil (1-\rho)N \rceil)}. \tag{20}$$

Note that $S_{(j)}$ is called the j-th *order-statistic* of the sequence $S(\boldsymbol{X}_1), \ldots, S(\boldsymbol{X}_N)$. Note also that $\hat{\gamma}_t$ is chosen such that the event $\{S(\boldsymbol{X}) \geq \hat{\gamma}_t\}$ is not too rare (it has a probability of around ρ), and therefore updating the reference parameter via a procedure such as (22) is not void of meaning.

2. **Adaptive updating of** $\mathbf{v}_t$. For fixed γ_t and $\mathbf{v}_{t-1}$, derive $\mathbf{v}_t$ from the solution of the following CE program

$$\max_{\mathbf{v}} D(\mathbf{v}) = \max_{\mathbf{v}} \mathbb{E}_{\mathbf{v}_{t-1}} I_{\{S(\boldsymbol{X}) \geq \gamma_t\}} W(\boldsymbol{X}; \boldsymbol{u}, \mathbf{v}_{t-1}) \ln f(\boldsymbol{X}; \mathbf{v}). \tag{21}$$

The stochastic counterpart of (21) is as follows: for fixed $\hat{\gamma}_t$ and $\hat{\mathbf{v}}_{t-1}$, derive $\hat{\mathbf{v}}_t$ from the solution of following program

$$\max_{\mathbf{v}} \hat{D}(\mathbf{v}) = \max_{\mathbf{v}} \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \hat{\gamma}_t\}} W(\boldsymbol{X}_i; \boldsymbol{u}, \hat{\mathbf{v}}_{t-1}) \ln f(\boldsymbol{X}_i; \mathbf{v}). \tag{22}$$

Thus, at the first iteration, starting with $\hat{\mathbf{v}}_0 = \boldsymbol{u}$, to get a good estimate for $\hat{\mathbf{v}}_1$, the target event is artificially made less rare by (temporarily) using a level $\hat{\gamma}_1$ which is chosen smaller than γ. The value for $\hat{\mathbf{v}}_1$ obtained in this way will (hopefully) make the event $\{S(\boldsymbol{X}) \geq \gamma\}$ less rare in the next iteration, so in the next iteration a value $\hat{\gamma}_2$ can be used which is closer to γ itself. The algorithm terminates when at some iteration t a level is reached which is at least γ and thus the original value of γ can be used without getting too few samples.

As mentioned before, the optimal solutions of (21) and (22) can often be obtained *analytically*, in particular when $f(\boldsymbol{x}; \mathbf{v})$ belongs to a NEF.

The above rationale results in the following algorithm.

Algorithm 2.1 (Main CE Algorithm for Rare Event Simulation)

1. Define $\hat{\mathbf{v}}_0 = \boldsymbol{u}$. Set $t = 1$ (iteration = level counter).
2. Generate a sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ from the density $f(\cdot; \mathbf{v}_{t-1})$ and compute the sample $(1-\rho)$-quantile $\hat{\gamma}_t$ of the performances according to (20), provided $\hat{\gamma}_t$ is less than γ. Otherwise set $\hat{\gamma}_t = \gamma$.
3. Use the **same** sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ to solve the stochastic program (22). Denote the solution by $\hat{\mathbf{v}}_t$.
4. If $\hat{\gamma}_t < \gamma$, set $t = t + 1$ and reiterate from step 2. Else proceed with step 5.
5. Estimate the rare-event probability ℓ using the LR estimate

$$\hat{\ell} = \frac{1}{N_1} \sum_{i=1}^{N_1} I_{\{S(\boldsymbol{X}_i) \geq \gamma\}} \, W(\boldsymbol{X}_i; \boldsymbol{u}, \hat{\mathbf{v}}_T), \tag{23}$$

where T denotes the final number of iterations (= number of levels used).

Example 2.1 We return to the example in Section 1.1. In this case, from (1) we have

$$\frac{\partial}{\partial v_j} \ln f(\boldsymbol{x}; \mathbf{v}) = \frac{x_j}{v_j^2} - \frac{1}{v_j},$$

so that the jth equation of (17) becomes

$$\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \gamma\}} \, W(\boldsymbol{X}_i; \boldsymbol{u}, \boldsymbol{w}) \left(\frac{X_{ij}}{v_j^2} - \frac{1}{v_j} \right) = 0, \quad j = 1, \ldots, 5;$$

therefore

$$v_j = \frac{\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \gamma\}} W(\boldsymbol{X}_i; \boldsymbol{u}, \boldsymbol{w}) \, X_{ij}}{\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \gamma\}} W(\boldsymbol{X}_i; \boldsymbol{u}, \boldsymbol{w})}, \tag{24}$$

which leads to the updating formula in step 3 of Algorithm 1.1. Actually, one can show that if the distributions belong to a natural exponential family, the updating formula always becomes (24).

We have intentionally used the notation $\hat{\gamma}_t$ and $\hat{\mathbf{v}}_t$ in Algorithm 2.1 above, rather than the more obvious γ_t and $\mathbf{v}_t$, in order to distinguish it from its *deterministic* counterpart, which is obtained by replacing sample means and sample quantiles by expectations and quantiles. For easy reference and better insight we present below the deterministic version of Algorithm 2.1. We have omitted the IS step in the algorithm below.

Algorithm 2.2 (Deterministic version of the CE algorithm)

1. Define $\mathbf{v}_0 := \boldsymbol{u}$. Set $t = 1$.
2. Calculate γ_t as

$$\gamma_t := \max\{s \ : \ \mathbb{P}_{\mathbf{v}_{t-1}}(S(\boldsymbol{X}) \geq s) \geq \rho\}, \tag{25}$$

   provided this is less than γ; otherwise put $\gamma_t := \gamma$.
3. Calculate $\mathbf{v}_t$ as

$$\mathbf{v}_t = \operatorname*{argmax}_{\mathbf{v}} \mathbb{E}_{\mathbf{v}_{t-1}} I_{\{S(\boldsymbol{X}) \geq \gamma_t\}} W(\boldsymbol{X}; \boldsymbol{u}, \mathbf{v}_{t-1}) \ln f(\boldsymbol{X}; \mathbf{v}). \tag{26}$$

4. If $\gamma_t = \gamma$, then **stop**; otherwise set $t := t + 1$ and reiterate from step 2.

Remark 2.1 (Static Simulation). The above method has been formulated for finite-dimensional random vectors only; this is sometimes referred to as *static* simulation. For infinite-dimensional random vectors or stochastic processes we need a more subtle treatment.

We will not go into details here, but the main point is that Algorithm 2.1 holds true without much alteration and can be readily applied to estimation problems involving both light and heavy tail distributions (de Boer, Kroese, and Rubinstein, 2004; Rubinstein and Kroese, 2004; Asmussen, Kroese, and Rubinstein, 2005).

Remark 2.2 (Variance Minimization). An alternative way to obtain a good reference parameter is to choose $\mathbf{v}$ such that the *variance*, or equivalently, the second moment, of the IS estimator is minimal. In other words we wish to find

$$_*\mathbf{v} = \operatorname*{argmin}_{\mathbf{v}} \mathbb{E}_{\mathbf{v}} \left[I_{\{S(\boldsymbol{X}) \geq \gamma\}} \, W(\boldsymbol{X}; \boldsymbol{u}, \mathbf{v}) \right]^2 . \tag{27}$$

More generally, using again the principle of importance sampling, this is equivalent to finding

$$_*\mathbf{v} = \operatorname*{argmin}_{\mathbf{v}} \mathbb{E}_{\boldsymbol{w}} \, I_{\{S(\boldsymbol{X}) \geq \gamma\}} \, W(\boldsymbol{X}; \boldsymbol{u}, \mathbf{v}) \, W(\boldsymbol{X}; \boldsymbol{u}, \boldsymbol{w}) \tag{28}$$

for *any* reference parameter $\boldsymbol{w}$. As in (16), we can estimate $_*\mathbf{v}$ as the solution to the stochastic program

$$\min_{\mathbf{v}} \hat{V}(\mathbf{v}) = \min_{\mathbf{v}} \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \gamma\}} \, W(\boldsymbol{X}_i; \boldsymbol{u}, \mathbf{v}) \, W(\boldsymbol{X}_i; \boldsymbol{u}, \boldsymbol{w}), \tag{29}$$

where $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ is a random sample from $f(\cdot; \boldsymbol{w})$. However, the evaluation of (29) in general involves complicated *numerical* optimization, and it is much more convenient to use the closed-form updating formulas that follow from CE minimization.

2.2. *The CE-method for combinatorial optimization*

In this subsection we discuss the main ideas behind the CE algorithm for combinatorial optimization. When reading this section, the reader is encouraged to refer back to the toy example in Section 1.2.

Consider the following general maximization problem. Let $\mathcal{X}$ be a finite set of *states*, and let S be a real-valued *performance function* on $\mathcal{X}$. We wish to find the maximum of S over $\mathcal{X}$, and the corresponding state(s) at which this maximum is attained. Let us denote the maximum by γ^*. Thus,

$$S(\boldsymbol{x}^*) = \gamma^* = \max_{\boldsymbol{x}\in\mathcal{X}} S(\boldsymbol{x}). \tag{30}$$

The starting point in the methodology of the CE method is to associate an *estimation problem* with the optimization problem (30). To this end we define a collection of indicator functions $\{I_{\{S(\boldsymbol{x})\geq\gamma\}}\}$ on $\mathcal{X}$ for various thresholds or *levels* $\gamma \in \mathbb{R}$. Next, let $\{f(\cdot;\mathbf{v}), \mathbf{v} \in \mathcal{V}\}$ be a family of (discrete) pdfs on $\mathcal{X}$, parameterized by a real-valued parameter (vector) $\mathbf{v}$.

For a certain $\boldsymbol{u} \in \mathcal{V}$ we associate with (30) the problem of estimating the number

$$\ell(\gamma) = \mathbb{P}_{\boldsymbol{u}}(S(\boldsymbol{X}) \geq \gamma) = \sum_{\boldsymbol{x}} I_{\{S(\boldsymbol{x})\geq\gamma\}} f(\boldsymbol{x};\boldsymbol{u}) = \mathbb{E}_{\boldsymbol{u}} I_{\{S(\boldsymbol{X})\geq\gamma\}}, \tag{31}$$

where $\mathbb{P}_{\boldsymbol{u}}$ is the probability measure under which the random state $\boldsymbol{X}$ has pdf $f(\cdot;\boldsymbol{u})$, and $\mathbb{E}_{\boldsymbol{u}}$ denotes the corresponding expectation operator. We will call the estimation problem (31) the *associated stochastic problem* (ASP). To indicate how (31) is associated with (30), suppose for example that γ is equal to γ^* and that $f(\cdot;\boldsymbol{u})$ is the uniform density on $\mathcal{X}$. Note that, typically, $\ell(\gamma^*) = f(\boldsymbol{x}^*;\boldsymbol{u}) = 1/|\mathcal{X}|$ – where $|\mathcal{X}|$ denotes the number of elements in $\mathcal{X}$ – is a very small number. Thus, for $\gamma = \gamma^*$ a natural way to estimate $\ell(\gamma)$ would be to use the LR estimator (23) with reference parameter $\mathbf{v}^*$ given by

$$\mathbf{v}^* = \operatorname*{argmax}_{\mathbf{v}} \mathbb{E}_{\boldsymbol{u}}\, I_{\{S(\boldsymbol{X})\geq\gamma\}} \ln f(\boldsymbol{X};\mathbf{v}). \tag{32}$$

This parameter could be estimated by

$$\hat{\mathbf{v}}^* = \operatorname*{argmax}_{\mathbf{v}} \frac{1}{N}\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i)\geq\gamma\}} \ln f(\boldsymbol{X}_i;\mathbf{v}), \tag{33}$$

where the $\boldsymbol{X}_i$ are generated from pdf $f(\cdot;\boldsymbol{u})$.

It is plausible that if γ is close to γ^* then $f(\cdot;\mathbf{v}^*)$ assigns most of its probability mass close to $\boldsymbol{x}^*$, and thus can be used to generate an approximate solution to (30). However, it is important to note that the estimator (33) is only of practical use when $I_{\{S(\boldsymbol{X})\geq\gamma\}} = 1$ for enough samples. This means for example that when γ is close to γ^*, $\boldsymbol{u}$ needs to be such that $\mathbb{P}_{\boldsymbol{u}}(S(\boldsymbol{X}) \geq \gamma)$ is not too small. Thus, the choice of $\boldsymbol{u}$ and γ in (31) are closely related. On the one hand we would like to choose γ as close as possible to γ^*, and find (an estimate of) $\mathbf{v}^*$ via the procedure above, which assigns almost all mass

to state(s) close to the optimal state. On the other hand, we would like to keep γ relative small in order to obtain an accurate (low RE) estimator for $\mathbf{v}^*$.

The situation is very similar to the rare event simulation case of Section 2.1. The idea, based essentially on Algorithm 2.1, is again to adopt a two-phase multi-level approach in which we simultaneously construct a *sequence* of levels $\hat{\gamma}_1, \hat{\gamma}_2, \ldots, \hat{\gamma}_T$ and parameter (vectors) $\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \ldots, \hat{\mathbf{v}}_T$ such that $\hat{\gamma}_T$ is close to the optimal γ^* and $\hat{\mathbf{v}}_T$ is such that the corresponding density assigns high probability mass to the collection of states that give a high performance.

This strategy is embodied in the following procedure, see e.g., Rubinstein (1999):

Algorithm 2.3 (Main CE Algorithm for Optimization)

1. Define $\hat{\mathbf{v}}_0 = \boldsymbol{u}$. Set $t = 1$ (level counter).
2. Generate a sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ from the density $f(\cdot; \mathbf{v}_{t-1})$ and compute the sample $(1-\rho)$-quantile $\hat{\gamma}_t$ of the performances according to (20).
3. Use the **same** sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ and solve the stochastic program (22) with $W = 1$. Denote the solution by $\hat{\mathbf{v}}_t$.
4. If for some $t \geq d$, say $d = 5$,

$$\hat{\gamma}_t = \hat{\gamma}_{t-1} = \cdots = \hat{\gamma}_{t-d}, \tag{34}$$

   then **stop** (let T denote the final iteration); otherwise set $t = t + 1$ and reiterate from step 2.

Note that the stopping criterion, the initial vector $\hat{\mathbf{v}}_0$, the sample size N and the number ρ have to be specified in advance, but that the rest of the algorithm is "self-tuning".

Remark 2.3 (Smoothed Updating). Instead of updating the parameter vector $\hat{\mathbf{v}}_{t-1}$ to $\hat{\mathbf{v}}_t$ directly via (33) we often use a *smoothed* updating procedure in which

$$\hat{\mathbf{v}}_t = \alpha\, \hat{\boldsymbol{w}}_t + (1-\alpha)\, \hat{\mathbf{v}}_{t-1}, \tag{35}$$

where $\hat{\boldsymbol{w}}_t$ is the vector derived via (22) with $W = 1$. This is especially relevant for optimization problems involving discrete random variables. The main reason why this heuristic smoothed updating procedure performs better is that it prevents the occurrences of 0s and 1s in the parameter vectors; once such an entry is 0 or 1, it often will remain so forever, which is undesirable. We found empirically that a value of α between $0.4 \leq \alpha \leq 0.9$ gives the best results. Clearly for $\alpha = 1$ we have the original updating rule in Algorithm 2.3.

In many applications we observed numerically that the sequence of pdfs $f(\cdot; \hat{\mathbf{v}}_0)$, $f(\cdot; \hat{\mathbf{v}}_1), \ldots$ converges to a degenerate measure (Dirac measure), assigning all probability mass to a single state $\boldsymbol{x}_T$, for which, by definition, the function value is greater than or equal to $\hat{\gamma}_T$.

Remark 2.4 (Similarities and Differences). Despite the great similarity between Algorithm 2.1 and Algorithm 2.3 it is important to note a number of differences. For example, the role of the initial reference parameter $\boldsymbol{u}$ is significantly different. In Algorithm 2.1 $\boldsymbol{u}$ is the unique *nominal* parameter for estimating $\mathbb{P}_{\boldsymbol{u}}(S(\boldsymbol{X}) \geq \gamma)$. However, in Algorithm 2.3 the choice for the initial parameter $\boldsymbol{u}$ is fairly arbitrary; it is only used to define the ASP. In contrast to Algorithm 2.1 the ASP for Algorithm 2.3 is redefined after each iteration. In particular, in steps 2 and 3 of Algorithm 2.3 we determine the optimal reference parameter associated with $\mathbb{P}_{\hat{\mathbf{v}}_{t-1}}(S(\boldsymbol{X}) \geq \hat{\gamma}_t)$, instead of $\mathbb{P}_{\boldsymbol{u}}(S(\boldsymbol{X}) \geq \hat{\gamma}_t)$. Consequently, the likelihood ratio term W that plays a crucial role in Algorithm 2.1 does not appear in Algorithm 2.3.

The above procedure can, in principle, be applied to any discrete and continuous (multi-extremal) optimization problem. However, for each individual problem two essential ingredients need to be supplied.

1. We need to specify how the samples are generated. In other words, we need to specify the family of pdfs $\{f(\cdot; \mathbf{v})\}$.
2. We need to calculate the updating rules for the parameters, based on cross-entropy minimization.

In general there are many ways to generate samples from $\mathcal{X}$, and it is not always immediately clear which way of generating the sample will yield better results or easier updating formulas.

Example 2.2. We return to the example from Section 1.2. In this case, the random vector $\boldsymbol{X} = (X_1, \ldots, X_n) \sim \mathsf{Ber}(\boldsymbol{p})$, and the parameter vector $\mathbf{v}$ is $\boldsymbol{p}$. Consequently, the pdf is

$$f(\boldsymbol{X}; \boldsymbol{p}) = \prod_{i=1}^{n} p_i^{X_i} (1 - p_i)^{1 - X_i},$$

and since each X_j can only be 0 or 1,

$$\frac{\partial}{\partial p_j} \ln f(\boldsymbol{X}; \boldsymbol{p}) = \frac{X_j}{p_j} - \frac{1 - X_j}{1 - p_j} = \frac{1}{(1 - p_j) p_j} (X_j - p_j).$$

Now we can find the maximum in (33) by setting the first derivatives w.r.t. p_j to zero, for $j = 1, \ldots, n$:

$$\frac{\partial}{\partial p_j} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \gamma\}} \ln f(\boldsymbol{X}_i; \boldsymbol{p}) = \frac{1}{(1 - p_j)\, p_j} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \gamma\}} (X_{ij} - p_j) = 0.$$

Thus, we get

$$p_j = \frac{\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i)\geq\gamma\}} X_{ij}}{\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i)\geq\gamma\}}}, \tag{36}$$

which immediately implies (8).

Several remarks are in order.

Remark 2.5 (Maximum Likelihood Estimation). It is interesting to note the connection between (33) and *maximum likelihood estimation* (MLE). In the MLE problem we are given data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ which are thought to be the outcomes of i.i.d. random variables $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ (random sample) each having a distribution $f(\cdot; \mathbf{v})$, where the parameter (vector) $\mathbf{v}$ is an element of some set V. We wish to estimate $\mathbf{v}$ on the basis of the data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$. The *maximum likelihood estimate* (MLE) is that parameter $\hat{\mathbf{v}}$ which maximizes the joint density of $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ for the given data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$. In other words,

$$\hat{\mathbf{v}} = \underset{\mathbf{v}}{\operatorname{argmax}} \prod_{i=1}^{N} f(\boldsymbol{x}_i; \mathbf{v}).$$

The corresponding random variable, obtained by replacing $\boldsymbol{x}_i$ with $\boldsymbol{X}_i$ is called the *maximum likelihood estimator* (MLE as well), also denoted by $\hat{\mathbf{v}}$. Since $\ln(\cdot)$ is an increasing function we have

$$\hat{\mathbf{v}} = \underset{\mathbf{v}}{\operatorname{argmax}} \sum_{i=1}^{N} \ln f(\boldsymbol{X}_i; \mathbf{v}). \tag{37}$$

Solving (37) is similar to solving (33). The only difference is the indicator function $I_{\{S(\boldsymbol{X}_i)\geq\gamma\}}$. We can write Step 3 in Algorithm 2.3 as

$$\hat{\mathbf{v}}_t = \underset{v}{\operatorname{argmax}} \sum_{\boldsymbol{X}_i : S(\boldsymbol{X}_i)\geq\hat{\gamma}_t} \ln f(\boldsymbol{X}_i; \mathbf{v}).$$

In other words, $\hat{\mathbf{v}}_t$ is equal to the MLE of $\hat{\mathbf{v}}_{t-1}$ based only on the vectors $\boldsymbol{x}_i$ in the random sample for which the performance is greater than or equal to $\hat{\gamma}_t$. For example, in Example 2.2 the MLE of p_j based on a random sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ is

$$\hat{p}_j = \frac{\sum_{i=1}^{N} X_{ij}}{N}.$$

Thus, if we base the MLE only on those vectors that have performance greater than or equal to γ, we obtain (36) immediately.

Remark 2.6 (Parameters). The choice for the sample size N and the parameter ρ depends on the size of the problem and the number of parameters in the ASP. In particular, for a SNN-type problem it is suggested to take the sample size as $N = c\,n$, where n is the number of *nodes* and c a constant ($c > 1$), say $5 \leq c \leq 10$. In contrast, for a SEN-type problem it is suggested to take $N = c\,n^2$, where n^2 is the number of *edges* in the network.

It is crucial to realize that the sample sizes $N = c\,n$ and $N = c\,n^2$ (with $c > 1$) are associated with the number of ASP parameters (n and n^2) that one needs to estimate for the SNN and SEN problems, respectively (see also the max-cut and the TSP examples below). Clearly, in order to reliably estimate k parameters, one needs to take *at least* a sample $N = c\,k$ for some constant $c > 1$. Regarding ρ, it is suggested to take ρ around 0.01, provided n is reasonably large, say $n \geq 100$; and it is suggested to take a larger ρ, say $\rho \approx \ln(n)/n$, for $n < 100$.

Alternatively, the choice of N and ρ can be determined *adaptively*. For example, in Homem-de-Mello and Rubinstein (2002) an adaptive algorithm is proposed that adjusts N automatically. The FACE algorithm of Section 4 is another option.

3. Various applications

In this section we discuss applications of the CE method to two typical COPs. We start in Section 3.1 with an SNN problem – the max-cut problem. We explain how to apply the CE method for this problem and provide two simple examples. In Section 3.2 we consider a typical SEN problem – the travelling salesman problem. We demonstrate the usefulness of the CE method and its fast convergence in a number of numerical examples. We further illustrate the dynamics of the CE method and show how fast the reference parameters converge.

3.1. The max-cut problem

The max-cut problem in a graph can be formulated as follows. Given a weighted graph $G(V, E)$ with node set $V = \{1, \ldots, n\}$ and edge set E, partition the nodes of the graph 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. We assume the weights are non-negative. We note that the max-cut problem is an NP-hard problem. Without loss of generality, we assume that the graph is complete. For simplicity we assume the graph is not directed. We can represent the possibly zero edge weights via a non-negative, symmetric *cost* matrix $C = (c_{ij})$ where c_{ij} denotes the weight of the edge from i to j.

Formally, a *cut* is a partition $\{V_1, V_2\}$ of V. For example, if $V = \{1, \ldots, 6\}$, then $\{\{1, 3, 4\}, \{2, 5, 6\}\}$ is a possible cut. The *cost* of a cut is the sum of the weights across the cut. As an example, consider the following 6×6 cost matrix

$$C = \begin{pmatrix} 0 & c_{12} & c_{13} & 0 & 0 & 0 \\ c_{21} & 0 & c_{23} & c_{24} & 0 & 0 \\ c_{31} & c_{32} & 0 & c_{34} & c_{35} & 0 \\ 0 & c_{42} & c_{43} & 0 & c_{45} & c_{46} \\ 0 & 0 & c_{53} & c_{54} & 0 & c_{56} \\ 0 & 0 & 0 & c_{64} & c_{65} & 0 \end{pmatrix} \tag{38}$$

corresponding to the graph in figure 3.

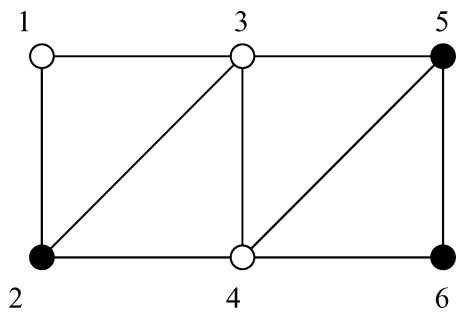

Figure 3. A 6-node network with the cut $\{\{1, 3, 4\}, \{2, 5, 6\}\}$.

For example, the cost of the cut $\{\{1, 3, 4\}, \{2, 5, 6\}\}$ is

$$c_{12} + c_{32} + c_{35} + c_{42} + c_{45} + c_{46}.$$

It will be convenient to represent a cut via a *cut vector* $\boldsymbol{x} = (x_1, \ldots, x_n)$, where $x_i = 1$ if node i belongs to the same partition as node 1, and 0 otherwise. By definition $x_1 = 1$. For example, the cut in figure 3 can be represented via the cut vector $(1, 0, 1, 1, 0, 0)$.

Let $\mathcal{X}$ be the set of all cut vectors $\boldsymbol{x} = (1, x_2, \ldots, x_n)$ and let $S(\boldsymbol{x})$ be the corresponding cost of the cut. We wish to maximize S via the CE method. Thus, (a) we need to specify how the random cut vectors are generated, and (b) calculate the corresponding updating formulas. The most natural and easiest way to generate the cut vectors is to let $X_2, \ldots, X_n$ be independent Bernoulli random variables with success probabilities $p_2, \ldots, p_n$, exactly as in the second toy example, see Section 1.2. It immediately follows, see Example 2.2, that the updating formula in Algorithm 2.3 at the tth iteration is given by

$$\hat{p}_{t,j} = \frac{\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \hat{\gamma}_t\}} \, I_{\{X_{ij}=1\}}}{\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \hat{\gamma}_t\}}}, \qquad j = 2, \ldots, n. \tag{39}$$

Remark 3.1 (r-partitions). We can readily extend the procedure to the case in which the node set V is partitioned into $r > 2$ subsets $\{V_1, \ldots, V_r\}$ such that the sum of the total weights of all edges going from the subset V_i to subset V_j, $i, j = 1, \ldots, r$, $(i \neq j)$ is maximized. In this case one can follow the basic steps of Algorithm 2.3 using independent r point distributions instead of independent Bernoulli distributions.

To illustrate the convergence of the CE algorithm we provide an example in which the exact optimal updating parameter is computed via (32) rather than estimated via (33). In other words, instead of Algorithm 2.3 its deterministic version Algorithm 2.2 is used (with $W = 1$ in (26)).

Example 3.1. Consider the 5-node graph with the cost matrix

$$C = \begin{pmatrix} 0 & 1 & 3 & 5 & 6 \\ 1 & 0 & 3 & 6 & 5 \\ 3 & 3 & 0 & 2 & 2 \\ 5 & 6 & 2 & 0 & 2 \\ 6 & 5 & 2 & 2 & 0 \end{pmatrix}. \tag{40}$$

It is readily seen that in this case the optimal cut vector is $\boldsymbol{x}^* = (1, 1, 0, 0, 0)$ with $S(\boldsymbol{x}^*) = \gamma^* = 28$.

We shall show next that in the "deterministic" version of Algorithm 2.3 the parameter vectors $\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots$ converge to the optimal $\boldsymbol{p}^* = (1, 1, 0, 0, 0)$ after two iterations, provided $\rho = 10^{-1}$ and $\boldsymbol{p}_0 = (1, 1/2, 1/2, 1/2, 1/2)$.

In the first step of the first iteration, we have to determine γ_1 from

$$\gamma_t = \max\left\{\gamma \;\text{ s.t. }\; \mathbb{E}_{\boldsymbol{p}_{t-1}} I_{\{S(\boldsymbol{X}) \geq \gamma\}} \geq 0.1\right\}. \tag{41}$$

Under parameter vector $\boldsymbol{p}_0$, $S(\boldsymbol{X})$ can take values, $\{0, 10, 15, 18, 19, 20, 21, 26, 28\}$ with probabilities $\{1, 1, 4, 2, 2, 2, 2, 1, 1\}/16$. Hence, we find $\gamma_1 = 26$. In the second step, we need to solve

$$\boldsymbol{p}_t = \underset{\boldsymbol{p}}{\operatorname{argmax}}\; \mathbb{E}_{\boldsymbol{p}_{t-1}} I_{\{S(\boldsymbol{X}) \geq \gamma_t\}} \ln f(\boldsymbol{X}; \boldsymbol{p}), \tag{42}$$

which has a solution similar to (36), namely

$$p_{t,j} = \frac{\mathbb{E}_{\boldsymbol{p}_{t-1}} I_{\{S(\boldsymbol{X}) \geq \gamma_t\}} X_j}{\mathbb{E}_{\boldsymbol{p}_{t-1}} I_{\{S(\boldsymbol{X}) \geq \gamma_t\}}}.$$

There are only two vectors $\boldsymbol{x}$ for which $S(\boldsymbol{x}) \geq 26$, namely $(1, 1, 1, 0, 0)$ and $(1, 1, 0, 0, 0)$, and both have probability 1/16 under $\boldsymbol{p}_0$. Thus,

$$p_{1,j} = \begin{cases} \dfrac{2/16}{2/16} = 1 & \text{for } j = 1, 2, \\ \dfrac{1/16}{2/16} = \dfrac{1}{2} & \text{for } j = 3, \\ \dfrac{0}{2/16} = 0 & \text{for } j = 4, 5. \end{cases}$$

In the second iteration $S(\boldsymbol{X})$ is 26 or 28 with probability 1/2. Thus, step (41) yields the (optimal) $\gamma_2 = 28$, and step (42) gives $\boldsymbol{p}_2 = (1, 1, 0, 0, 0)$.

Example 3.2 (A synthetic max-cut problem)**.** Since the max-cut problem is NP hard (Garey and Johnson, 1979; Papadimitriou and Yannakakis, 1991), no efficient method

for solving the max-cut problem exists. The naive total enumeration routine is only feasible for small graphs, say for those with $n \leq 30$ nodes. Although the Branch-and-Bound heuristic can solve medium size problems exactly, it too will run into problems when n becomes large.

In order to verify the accuracy of the CE method we construct an artificial graph such that the solution is available in advance. In particular, for $m \in \{1, \ldots, n\}$ consider the following symmetric cost matrix:

$$C = \begin{pmatrix} Z_{11} & B_{12} \\ B_{21} & Z_{22} \end{pmatrix}, \tag{43}$$

where Z_{11} is an $m \times m$ symmetric matrix in which all the upper-diagonal elements are generated from a $\mathsf{U}(a, b)$ distribution (and all lower-diagonal elements follow by symmetry), Z_{22} is a $(n-m) \times (n-m)$ symmetric matrix which is generated in a similar way as Z_{11}, and all the other elements are c, apart from the diagonal elements, which are of course 0.

It is not difficult to see that for $c > b(n-m)/m$, by construction, the optimal cut is given by $V^* = \{V_1^*, V_2^*\}$, with

$$V_1^* = \{1, \ldots, m\} \quad \text{and} \quad V_2^* = \{m+1, \ldots, n\}, \tag{44}$$

and the optimal value of the cut is

$$\gamma^* = c\, m\, (n-m).$$

Of course the same optimal solution and optimal value can be found for the general case where the elements in Z_{11} and Z_{22} are generated via an arbitrary bounded support distribution with the maximal value of the support less than b.

Table 3 lists a typical output of Algorithm 2.3 applied to the synthetic max-cut problem, for a network with $n = 400$ nodes. In this table we list besides the $(1-\rho)$-quantile of the performances $\hat{\gamma}_t$ also the *best* of the performances in each iteration, denoted by $S_{t,(N)}$, and the Euclidean distance

$$\|\hat{\boldsymbol{p}}_t - \boldsymbol{p}^*\| = \sqrt{(\hat{p}_{t,i} - p_i^*)^2},$$

as a measure how close the reference vector is to the optimal reference vector $\boldsymbol{p}^* = (1, 1, \ldots, 1, 0, 0, \ldots, 0)$.

In this particular example, $m = 200$ and the Z_{11} and Z_{22} are generated from the $\mathsf{U}(0, 1)$ distribution. The elements in B_{12} and B_{21} are constant $c = 1$. The CE parameter are chosen as follows: rarity parameter $\rho = 0.1$; smoothing parameter $\alpha = 1.0$ (no smoothing); stopping constant $d = 3$; and number of samples per iteration $N = 1000$.

The CPU time is only 100 seconds, using a Matlab implementation on a pentium III, 500 MHz processor. We see that the CE algorithm converges quickly, yielding the exact optimal solution 40000 in less than 23 iterations.

Table 3
A typical performance of Algorithm 2.3 for the synthetic max-cut problem with $n = 400$, $d = 3$, $\rho = 0.1$, $\alpha = 1.0$, $N = 1000$.

t	$\hat{\gamma}_t$	$S_{t,(N)}$	$\|\hat{\boldsymbol{p}}_t - \boldsymbol{p}^*\|$
1	30085.3	30320.9	9.98
2	30091.3	30369.4	10.00
3	30113.7	30569.8	9.98
4	30159.2	30569.8	9.71
5	30350.8	30652.9	9.08
6	30693.6	31244.7	8.37
7	31145.1	31954.8	7.65
8	31711.8	32361.5	6.94
9	32366.4	33050.3	6.27
10	33057.8	33939.9	5.58
11	33898.6	34897.9	4.93
12	34718.9	35876.4	4.23
13	35597.1	36733.0	3.60
14	36368.9	37431.7	3.02
15	37210.5	38051.2	2.48
16	37996.7	38654.5	1.96
17	38658.8	39221.9	1.42
18	39217.1	39707.8	1.01
19	39618.3	40000.0	0.62
20	39904.5	40000.0	0.29
21	40000.0	40000.0	0.14
22	40000.0	40000.0	0.00
23	40000.0	40000.0	0.00

Figures 4 and 5 illustrate the convergence of the reference vectors $\boldsymbol{p}_t$ to the optimal $\boldsymbol{p}^*$.

3.2. The travelling salesman problem

The travelling salesman problem (TSP) can be formulated as follows. Consider a weighted graph G with n nodes, labelled $1, 2, \ldots, n$. The nodes represent cities, and the edges represent the roads between the cities. Each edge from i to j has weight or cost c_{ij}, representing the length of the road. The problem is to find the shortest *tour* that visits all the cities exactly once[2] (except the starting city, which is also the terminating city), see figure 6.

Without loss of generality, let us assume that the graph is *complete*, and that some of the weights may be $+\infty$. Let $\mathcal{X}$ be the set of all possible tours and let $S(\boldsymbol{x})$ the total length

[2] In some versions of the TSP cities can be visited more than once.

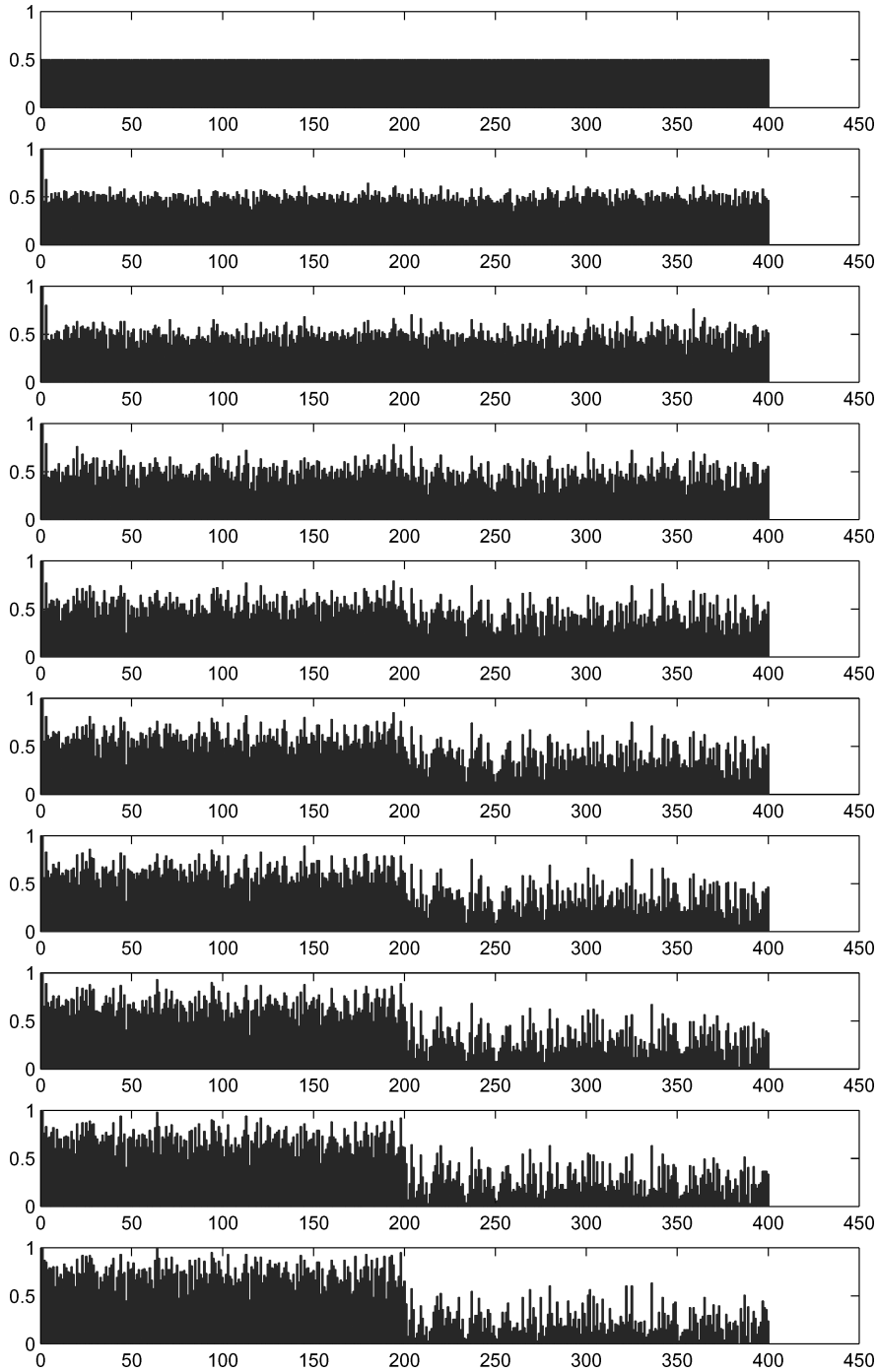

Figure 4. Sequence of reference vectors for the synthetic max-cut problem with 400 nodes. Iterations 0,1,. . . ,9.

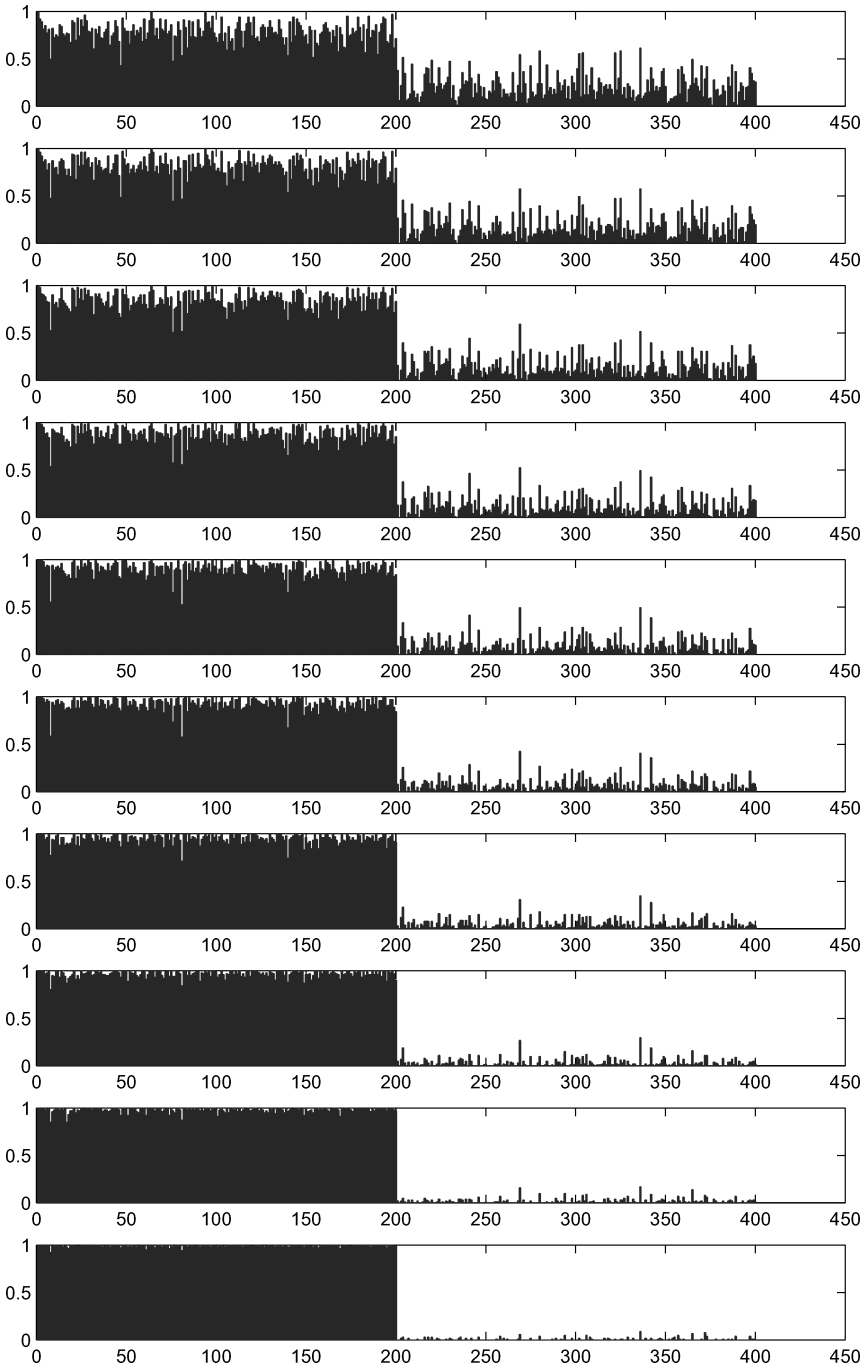

Figure 5. Sequence of reference vectors for the synthetic max-cut problem with 400 nodes. Iterations 10,. . . , 19.

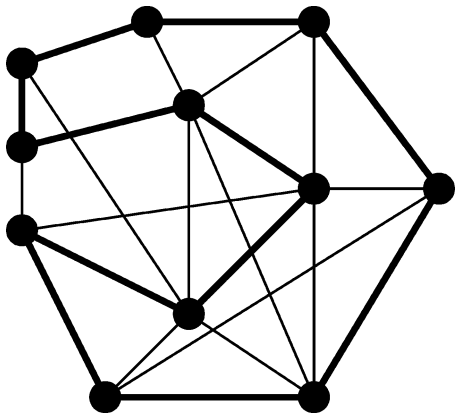

Figure 6. Find the shortest tour $\boldsymbol{x}$ visiting all nodes.

of tour $\boldsymbol{x} \in \mathcal{X}$. We can represent each tour via a *permutation* of $(1, \ldots, n)$. For example for $n = 4$, the permutation $(1, 3, 2, 4)$ represents the tour $1 \to 3 \to 2 \to 4 \to 1$. In fact, we may as well represent a tour via a permutation $\boldsymbol{x} = (x_1, \ldots, x_n)$ with $x_1 = 1$. From now on we identify a tour with its corresponding permutation, where $x_1 = 1$. We may now formulate the TSP as follows:

$$\min_{\boldsymbol{x} \in \mathcal{X}} S(\boldsymbol{x}) = \min_{\boldsymbol{x} \in \mathcal{X}} \left\{ \sum_{i=1}^{n-1} c_{x_i, x_{i+1}} + c_{x_n, 1} \right\}. \tag{45}$$

Note that the number of elements in $\mathcal{X}$ is typically very large:

$$|\mathcal{X}| = (n-1)! \tag{46}$$

This is exactly the setting of Section 2.2, so we can use the CE-Method to solve (45). Note however that we need to modify Algorithm 2.3 since we have here a *minimization* problem.

In order to apply the CE algorithm we need to specify (a) how to generate the random tours, and (b) how to update the parameters at each iteration.

The easiest way to explain how the tours are generated and how the parameters are updated is to relate (45) to an *equivalent* minimization problem. Let

$$\tilde{\mathcal{X}} = \{(x_1, \ldots, x_n) \,:\, x_1 = 1, x_i \in \{1, \ldots, n\}, i = 2, \ldots, n\},$$

be the set of vectors that correspond to paths that start and end in 1 and can visit the same city more than once. Note that $|\tilde{\mathcal{X}}| = n^{n-1}$, and $\mathcal{X} \subset \tilde{\mathcal{X}}$. When $n = 4$, we have for example $\boldsymbol{x} = (1, 3, 1, 3) \in \tilde{\mathcal{X}}$, corresponding to the path $1 \to 3 \to 1 \to 3 \to 1$. Again we will identify the vectors with their corresponding paths. Define the function $\tilde{S}$ on $\tilde{\mathcal{X}}$ by $\tilde{S}(\boldsymbol{x}) = S(\boldsymbol{x})$, if $\boldsymbol{x} \in \mathcal{X}$ and $\tilde{S}(\boldsymbol{x}) = \infty$, otherwise. Then, obviously (45) is equivalent

to the minimization problem

$$\text{minimize } \tilde{S}(\boldsymbol{x}) \text{ over } \boldsymbol{x} \in \tilde{\mathcal{X}}. \tag{47}$$

A simple method to generate a random path $\boldsymbol{X} = (X_1, \ldots, X_n)$ in $\tilde{\mathcal{X}}$ is to use a Markov chain of the graph G, starting at node 1, and stopping after n steps. Let $P = (p_{ij})$ denote the one-step transition matrix of this Markov chain. We assume that the diagonal elements of P are 0, and that all other elements of P are strictly positive, but otherwise P is a general $n \times n$ stochastic matrix.

The pdf $f(\cdot; P)$ of $\boldsymbol{X}$ is thus parameterized by the matrix P and its logarithm is given by

$$\ln f(\boldsymbol{x}; P) = \sum_{r=1}^{n} \sum_{i,j} I_{\{\boldsymbol{x} \in \tilde{\mathcal{X}}_{ij}(r)\}} \ln p_{ij},$$

where $\tilde{\mathcal{X}}_{ij}(r)$ is the set of all paths in $\tilde{\mathcal{X}}$ for which the rth transition is from node i to j. The updating rules for this modified optimization problem follow from 22 ($W = 1$), with $\{S(\boldsymbol{X}_i) \geq \gamma\}$ replaced with $\{\tilde{S}(\boldsymbol{X}_i) \leq \gamma\}$, under the condition that the rows of P sum up to 1. Using Lagrange multipliers $u_1, \ldots, u_n$ we obtain the maximization problem

$$\max_P \min_u \left[\mathbb{E}_P I_{\{\tilde{S}(\boldsymbol{X}) \leq \gamma\}} \ln f(\boldsymbol{X}; P) + \sum_{i=1}^{n} u_i \left(\sum_{j=1}^{n} p_{ij} - 1 \right) \right].$$

Differentiating the expression within square brackets above with respect to p_{ij}, yields, for all $j = 1, \ldots, n$,

$$\frac{\mathbb{E}_P I_{\{\tilde{S}(\boldsymbol{X}) \leq \gamma\}} \sum_{r=1}^{n} I_{\{\boldsymbol{X} \in \tilde{\mathcal{X}}_{ij}(r)\}}}{p_{ij}} + u_i = 0.$$

Summing over $j = 1, \ldots, n$ gives $\mathbb{E}_P I_{\{\tilde{S}(\boldsymbol{X}) \leq \gamma\}} \sum_{r=1}^{n} I_{\{\boldsymbol{X} \in \tilde{\mathcal{X}}_i(r)\}} = -u_i$, where $\tilde{\mathcal{X}}_i(r)$ is the set of paths for which the r-th transition starts from node i. It follows that the optimal p_{ij} is given by

$$p_{ij} = \frac{\mathbb{E}_P I_{\{\tilde{S}(\boldsymbol{X}) \leq \gamma\}} \sum_{r=1}^{n} I_{\{\boldsymbol{X} \in \tilde{\mathcal{X}}_{ij}(r)\}}}{\mathbb{E}_P I_{\{\tilde{S}(\boldsymbol{X}) \leq \gamma\}} \sum_{r=1}^{n} I_{\{\boldsymbol{X} \in \tilde{\mathcal{X}}_i(r)\}}}.$$

The corresponding estimator is

$$\hat{p}_{ij} = \frac{\sum_{k=1}^{N} I_{\{\tilde{S}(\boldsymbol{X}_k) \leq \gamma\}} \sum_{r=1}^{n} I_{\{\boldsymbol{X}_k \in \tilde{\mathcal{X}}_{ij}(r)\}}}{\sum_{k=1}^{N} I_{\{\tilde{S}(\boldsymbol{X}_k) \leq \gamma\}} \sum_{r=1}^{n} I_{\{\boldsymbol{X}_k \in \tilde{\mathcal{X}}_i(r)\}}}. \tag{48}$$

This has a very simple interpretation. To update p_{ij} we simply take the fraction of times that the transitions from i to j occurred, taking only those paths into account that have a total length less than or equal to γ.

This is how we could, *in principle*, carry out the sample generation and parameter updating for problem (47). We generate the path via a Markov process with transition matrix P, and use updating formula (48). However, *in practice*, we would never generate the paths in this way, since the majority of these paths would be irrelevant since they would not constitute a tour, and therefore their $\tilde{S}$ values would be ∞. In order to avoid the generation of irrelevant paths, we proceed as follows.

Algorithm 3.1 (Generation of permutations (tours) in the TSP)

1. Define $P^{(1)} = P$ and $X_1 = 1$. Let $k = 1$.
2. Obtain $P^{(k+1)}$ from $P^{(k)}$ by first setting the X_k-th column of $P^{(k)}$ to 0 and then normalizing the rows to sum up to 1. Generate X_{k+1} from the distribution formed by the X_k-th row of $P^{(k)}$.
3. If $k = n - 1$ then **stop**; otherwise set $k = k + 1$ and reiterate from step 2.

It is important to realize that the updating formula remains the same; by using Algorithm 3.1 we are merely *speeding-up* our naive way of generating the tours. Moreover, since we now only generate *tours*, the updated value for p_{ij} can be estimated as

$$\hat{p}_{ij} = \frac{\sum_{k=1}^{N} I_{\{S(\boldsymbol{X}_k)\leq\gamma\}}\, I_{\{\boldsymbol{X}_k\in\mathcal{X}_{ij}\}}}{\sum_{k=1}^{N} I_{\{S(\boldsymbol{X}_k)\leq\gamma\}}}, \tag{49}$$

where $\mathcal{X}_{ij}$ is the set of tours in which the transition from i to j is made. This has the same "natural" interpretation as discussed for (48).

To complete the algorithm, we need to specify the initialization conditions and the stopping criterion. For the initial matrix $\hat{P}_0$ we could simply take all off-diagonal elements equal to $1/(n-1)$ and for the stopping criterion use formula (34).

Numerical Examples

To demonstrate the usefulness of the CE algorithm and its fast and accurate convergence we provide a number of numerical examples. The first example concerns the benchmark TSP problem `ft53` taken from the URL

http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/atsp/

Table 4
A typical performance of Algorithm 2.3 for the TSP problem `ft53` with $n = 53$ nodes, $d = 5$, $\rho = 0.01$, $\alpha = 0.7$, $N = 10\,n^2 = 28090$.

t	$\hat{\gamma}_t$	$S_{t,(1)}$	P_t^{mm}	fdiff
1	23234.00	21111.00	0.0354	0.3243
2	20611.00	18586.00	0.0409	0.3744
3	18686.00	16819.00	0.0514	0.3707
4	17101.00	14890.00	0.0465	0.3676
5	15509.00	13459.00	0.0698	0.3635
6	14449.00	12756.00	0.0901	0.3492
7	13491.00	11963.00	0.0895	0.3404
8	12773.00	11326.00	0.1065	0.3250
9	12120.00	10357.00	0.0965	0.3291
10	11480.00	10216.00	0.1034	0.3167
11	11347.00	9952.00	0.1310	0.3017
12	10791.00	9525.00	0.1319	0.3111
13	10293.00	9246.00	0.1623	0.3049
14	10688.00	9176.00	0.1507	0.2726
15	9727.00	8457.00	0.1346	0.3203
16	9263.00	8424.00	0.1436	0.2929
17	9422.00	8614.00	0.1582	0.2490
18	9155.00	8528.00	0.1666	0.2468
19	8661.00	7970.00	0.1352	0.2543
20	8273.00	7619.00	0.1597	0.2360
21	8096.00	7485.00	0.1573	0.2163
22	7868.00	7216.00	0.1859	0.1889
23	7677.00	7184.00	0.2301	0.1737
24	7519.00	7108.00	0.2421	0.1569
25	7420.00	7163.00	0.2861	0.1495
26	7535.00	7064.00	0.3341	0.1508
27	7506.00	7072.00	0.3286	0.1576
28	7199.00	7008.00	0.3667	0.1341
29	7189.00	7024.00	0.3487	0.1212
30	7077.00	7008.00	0.4101	0.0998
31	7068.00	7008.00	0.4680	0.1051

Table 4 presents the performance of the CE Algorithm for the problem `ft53`, which defines an asymmetric fully connected graph of size 53, where the value of each edge c_{ij} is given.
The CE parameters were: stopping parameter $d = 5$, rarity parameter $\rho = 0.01$, sample size $N = 10\,n^2 = 28090$ and smoothing parameter $\alpha = 0.7$. The relative experimental error of the solution is

$$\varepsilon = \frac{\hat{\gamma}_T - \gamma^*}{\gamma^*} = 1.5\%,$$

Table 5
Case studies for the TSP.

File	n	$\bar{T}$	$\bar{\gamma}_1$	$\bar{\gamma}_T$	γ^*	$\bar{\varepsilon}$	ε_*	ε^*	CPU
br17	17	23.8	68.2	39.0	39	0.0	0.0	0.0	9
ftv33	34	31.2	3294.0	1312.2	1286	2.0	0.0	6.2	73
ftv35	36	31.5	3714.0	1490.0	1473	1.2	0.4	1.8	77
ftv38	39	33.8	4010.8	1549.8	1530	1.3	0.4	3.2	132
p43	43	44.5	9235.5	5624.5	5620	0.1	0.0	0.1	378
ftv44	45	35.5	4808.2	1655.8	1613	2.7	1.3	3.3	219
ftv47	48	40.2	5317.8	1814.0	1776	2.1	0.6	4.1	317
ry48p	48	40.8	40192.0	14845.5	14422	2.9	1.9	5.0	345
ft53	53	39.5	20889.5	7103.2	6905	2.9	2.5	3.5	373
ftv55	56	40.0	5835.8	1640.0	1608	2.0	0.2	4.3	408
ftv64	65	43.2	6974.2	1850.0	1839	0.6	0.0	1.4	854
ftv70	71	47.0	7856.8	1974.8	1950	1.3	0.4	3.7	1068
ft70	70	42.8	64199.5	39114.8	38673	1.1	0.3	1.9	948

where $\gamma^* = 6905$ is the best known solution. The CPU time was approximately 6 minutes. In Table 4 $S_{t,(1)}$ denotes the length of smallest tour in iteration t. We also included the two quantities P_t^{mm} and fdiff, which give extra information about the dynamics. Specifically, fdiff is the proportion, out of N, of different values for the performance function, for the current iteration t; and P_t^{mm} is the minimum of the maximum elements in each row of matrix $\hat{P}_t$.

Similar performances were found for other TSP problems in the benchmark library above. Table 5 presents the performance of Algorithm 2.3 for a selection of case studies from this library. In all numerical results we use the same CE parameters as for the ft53 problem, that is $\rho = 10^{-2}$, $N = 10n^2$, $\alpha = 0.7$ (smoothing parameter in (35)) and $d = 5$ (in (34)). To study the variability in the solutions, each problem was repeated 10 times. In Table 5 n denotes the number of nodes of the graph, $\bar{T}$ denotes the average total number of iterations needed before stopping, $\bar{\gamma}_1$ and $\bar{\gamma}T$ denote the average initial and average final estimates of the optimal solution, γ^* denotes the best known (optimal) solution, $\bar{\varepsilon}$ denotes the average percentage relative experimental error based on 10 replications, ε_* and ε^* denote the smallest and the largest percentage relative error among the 10 generated solutions, and finally CPU denotes the average CPU time in seconds.

We found that decreasing the sample size N from $N = 10n^2$ to $N = 5n^2$ all relative experimental errors ε in Table 5 increase at most by a factor of 1.5.

Dynamics

Finally, as an illustration of the dynamics of the CE algorithm, we display below the sequence of matrices $\hat{P}_0, \hat{P}_1, \ldots$ for a TSP with $n = 10$ cities, where the optimal tour is

(1, 2, 3, . . . , 10, 1).

$$\hat{P}_0 = \begin{pmatrix} 0.00 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 \\ 0.11 & 0.00 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.00 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 & 0.00 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 & 0.11 & 0.00 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.00 & 0.11 & 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.00 & 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.00 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.00 & 0.11 \\ 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.00 \end{pmatrix}.$$

$$\hat{P}_1 = \begin{pmatrix} 0.00 & 0.31 & 0.04 & 0.08 & 0.04 & 0.19 & 0.08 & 0.08 & 0.12 & 0.08 \\ 0.04 & 0.00 & 0.33 & 0.08 & 0.17 & 0.08 & 0.08 & 0.04 & 0.04 & 0.12 \\ 0.08 & 0.08 & 0.00 & 0.23 & 0.04 & 0.04 & 0.12 & 0.19 & 0.08 & 0.15 \\ 0.12 & 0.19 & 0.08 & 0.00 & 0.12 & 0.08 & 0.08 & 0.08 & 0.19 & 0.08 \\ 0.08 & 0.08 & 0.19 & 0.08 & 0.00 & 0.23 & 0.08 & 0.04 & 0.15 & 0.08 \\ 0.04 & 0.04 & 0.08 & 0.04 & 0.12 & 0.00 & 0.50 & 0.08 & 0.08 & 0.04 \\ 0.23 & 0.08 & 0.08 & 0.04 & 0.08 & 0.04 & 0.00 & 0.27 & 0.08 & 0.12 \\ 0.08 & 0.15 & 0.04 & 0.04 & 0.19 & 0.08 & 0.08 & 0.00 & 0.27 & 0.08 \\ 0.08 & 0.08 & 0.04 & 0.12 & 0.08 & 0.15 & 0.08 & 0.04 & 0.00 & 0.35 \\ 0.21 & 0.08 & 0.17 & 0.08 & 0.04 & 0.12 & 0.08 & 0.12 & 0.08 & 0.00 \end{pmatrix}.$$

$$\hat{P}_2 = \begin{pmatrix} 0.00 & 0.64 & 0.03 & 0.06 & 0.04 & 0.04 & 0.06 & 0.04 & 0.04 & 0.06 \\ 0.03 & 0.00 & 0.58 & 0.07 & 0.07 & 0.05 & 0.05 & 0.03 & 0.03 & 0.08 \\ 0.05 & 0.05 & 0.00 & 0.52 & 0.04 & 0.03 & 0.08 & 0.04 & 0.05 & 0.15 \\ 0.04 & 0.13 & 0.05 & 0.00 & 0.22 & 0.18 & 0.05 & 0.04 & 0.25 & 0.05 \\ 0.06 & 0.04 & 0.09 & 0.04 & 0.00 & 0.60 & 0.04 & 0.03 & 0.04 & 0.06 \\ 0.03 & 0.03 & 0.05 & 0.03 & 0.04 & 0.00 & 0.71 & 0.05 & 0.05 & 0.03 \\ 0.20 & 0.04 & 0.05 & 0.03 & 0.05 & 0.03 & 0.00 & 0.51 & 0.05 & 0.04 \\ 0.05 & 0.08 & 0.03 & 0.04 & 0.23 & 0.05 & 0.05 & 0.00 & 0.42 & 0.05 \\ 0.05 & 0.05 & 0.04 & 0.07 & 0.07 & 0.10 & 0.05 & 0.03 & 0.00 & 0.54 \\ 0.50 & 0.05 & 0.04 & 0.05 & 0.04 & 0.08 & 0.05 & 0.14 & 0.05 & 0.00 \end{pmatrix}.$$

$$\hat{P}_3 = \begin{pmatrix} 0.00 & 0.76 & 0.02 & 0.04 & 0.03 & 0.03 & 0.04 & 0.03 & 0.03 & 0.04 \\ 0.02 & 0.00 & 0.73 & 0.05 & 0.05 & 0.04 & 0.04 & 0.02 & 0.02 & 0.05 \\ 0.03 & 0.03 & 0.00 & 0.70 & 0.02 & 0.02 & 0.05 & 0.02 & 0.03 & 0.09 \\ 0.02 & 0.07 & 0.03 & 0.00 & 0.59 & 0.10 & 0.03 & 0.02 & 0.13 & 0.03 \\ 0.04 & 0.03 & 0.06 & 0.03 & 0.00 & 0.73 & 0.03 & 0.02 & 0.03 & 0.04 \\ 0.02 & 0.02 & 0.04 & 0.02 & 0.03 & 0.00 & 0.79 & 0.04 & 0.04 & 0.02 \\ 0.12 & 0.02 & 0.03 & 0.02 & 0.03 & 0.02 & 0.00 & 0.69 & 0.03 & 0.02 \\ 0.03 & 0.05 & 0.02 & 0.02 & 0.14 & 0.03 & 0.03 & 0.00 & 0.66 & 0.03 \\ 0.03 & 0.03 & 0.02 & 0.05 & 0.05 & 0.06 & 0.03 & 0.02 & 0.00 & 0.71 \\ 0.69 & 0.03 & 0.02 & 0.03 & 0.02 & 0.05 & 0.03 & 0.09 & 0.03 & 0.00 \end{pmatrix}.$$

$$\hat{P}_4 = \begin{pmatrix} 0.00 & 0.82 & 0.01 & 0.03 & 0.02 & 0.02 & 0.03 & 0.02 & 0.02 & 0.03 \\ 0.01 & 0.00 & 0.80 & 0.03 & 0.03 & 0.03 & 0.03 & 0.01 & 0.01 & 0.04 \\ 0.02 & 0.02 & 0.00 & 0.79 & 0.02 & 0.01 & 0.03 & 0.02 & 0.02 & 0.07 \\ 0.01 & 0.04 & 0.02 & 0.00 & 0.73 & 0.06 & 0.02 & 0.01 & 0.09 & 0.02 \\ 0.03 & 0.02 & 0.04 & 0.02 & 0.00 & 0.81 & 0.02 & 0.01 & 0.02 & 0.03 \\ 0.01 & 0.01 & 0.03 & 0.01 & 0.02 & 0.00 & 0.84 & 0.03 & 0.03 & 0.01 \\ 0.09 & 0.02 & 0.02 & 0.01 & 0.02 & 0.01 & 0.00 & 0.78 & 0.02 & 0.02 \\ 0.02 & 0.03 & 0.01 & 0.02 & 0.09 & 0.02 & 0.02 & 0.00 & 0.76 & 0.02 \\ 0.02 & 0.02 & 0.02 & 0.03 & 0.03 & 0.05 & 0.02 & 0.01 & 0.00 & 0.79 \\ 0.78 & 0.02 & 0.02 & 0.02 & 0.02 & 0.03 & 0.02 & 0.06 & 0.02 & 0.00 \end{pmatrix}.$$

$$\hat{P}_5 = \begin{pmatrix} 0.00 & 0.86 & 0.01 & 0.02 & 0.02 & 0.02 & 0.02 & 0.02 & 0.02 & 0.02 \\ 0.01 & 0.00 & 0.85 & 0.03 & 0.03 & 0.02 & 0.02 & 0.01 & 0.01 & 0.03 \\ 0.02 & 0.02 & 0.00 & 0.84 & 0.01 & 0.01 & 0.03 & 0.01 & 0.02 & 0.05 \\ 0.01 & 0.03 & 0.01 & 0.00 & 0.80 & 0.05 & 0.01 & 0.01 & 0.06 & 0.01 \\ 0.02 & 0.02 & 0.03 & 0.02 & 0.00 & 0.85 & 0.02 & 0.01 & 0.02 & 0.02 \\ 0.01 & 0.01 & 0.02 & 0.01 & 0.02 & 0.00 & 0.88 & 0.02 & 0.02 & 0.01 \\ 0.06 & 0.01 & 0.02 & 0.01 & 0.02 & 0.01 & 0.00 & 0.84 & 0.02 & 0.01 \\ 0.02 & 0.02 & 0.01 & 0.01 & 0.07 & 0.02 & 0.02 & 0.00 & 0.82 & 0.02 \\ 0.02 & 0.02 & 0.01 & 0.02 & 0.02 & 0.03 & 0.02 & 0.01 & 0.00 & 0.84 \\ 0.84 & 0.02 & 0.01 & 0.02 & 0.01 & 0.03 & 0.02 & 0.05 & 0.02 & 0.00 \end{pmatrix}.$$

An illustration of the dynamics is given in figure 7.

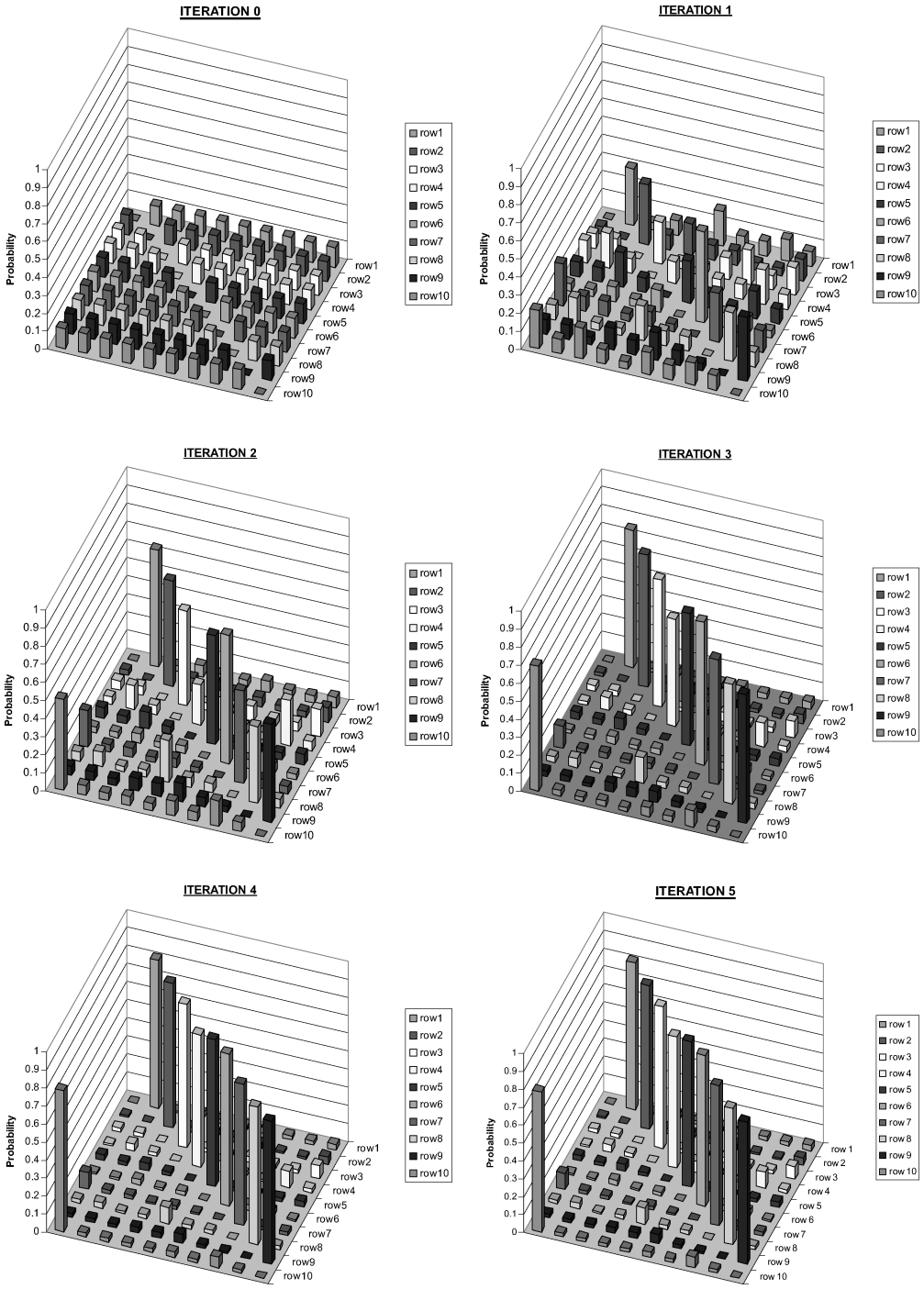

Figure 7. Convergence of the reference parameter (matrix) for a 10 node TSP.

4. Modifications

In this section we consider several modifications of the basic CE algorithm. We start by discussing alternative loss functions that favor samples with higher performance more than other samples. We continue with a fully automated version of the CE algorithm. This modified version allows automatic tuning of all the parameters of the CE algorithm.

4.1. Alternative performance functions

Consider the maximization problem, see (30),

$$\max_{\boldsymbol{x}\in\mathcal{X}} S(\boldsymbol{x}),$$

where $S(\boldsymbol{x})$ is some positive objective function defined on $\mathcal{X}$. In the associated stochastic problem, see (31), we consider instead estimating the rare-event probability

$$\ell(\gamma) = \mathbb{E}_{\boldsymbol{u}} I_{\{S(\boldsymbol{X})\geq\gamma\}}.$$

Note that this can be written as

$$\ell(\gamma) = \mathbb{E}_{\boldsymbol{u}} \varphi(S(\boldsymbol{X});\gamma),$$

where $\varphi(s;\gamma)$ is the indicator $I_{\{s\geq\gamma\}}$, that is,

$$\varphi(s;\gamma) = \begin{cases} 1 & \text{if } s \geq \gamma, \\ 0 & \text{if } s < \gamma, \end{cases} \tag{50}$$

for $s \geq 0$. Recall that the main CE Algorithm 2.3 contains the steps (a) updating $\hat{\gamma}_t$ via (20) and (b) updating $\hat{\mathbf{v}}_t$ via the (analytic) solution of (22). A natural modification of Algorithm 2.3 would be to update $\mathbf{v}_t$ using an alternative function $\varphi(s;\gamma)$. For a maximization problem such a function should be increasing in s for each fixed $\gamma \geq 0$, and decreasing in γ for each fixed $s \geq 0$. In particular one could use

$$\varphi(s;\gamma) = \psi(s) I_{\{s\geq\gamma\}},$$

for some increasing function $\psi(s)$.

Using such a $\varphi(s;\gamma)$ instead of the indicator (50) we now proceed similar as before. Specifically, the updating step (a) of $\hat{\gamma}_t$ remains *exactly the same*, and the updating step (b) of $\hat{\mathbf{v}}_t$ now reduces to the solution of the following program

$$\max_{\mathbf{v}} \hat{D}(\mathbf{v}) = \max_{\mathbf{v}} \frac{1}{N}\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i)\geq\hat{\gamma}_t\}}\, \psi(S(\boldsymbol{X}_i))\, \ln f(\boldsymbol{X}_i;\mathbf{v}). \tag{51}$$

Numerical evidence suggests that $\psi(s) = s$ can lead to some speed-up of the algorithm. As an example, for the max-cut problem we obtain in this case (see (39)) the analytic updating formulas

$$\hat{p}_{t,i} = \frac{\sum_{k=1}^{N} I_{\{S(\boldsymbol{X}_k) \geq \hat{\gamma}_t\}}\, S(\boldsymbol{X}_i)\, X_{ki}}{\sum_{k=1}^{N} I_{\{S(\boldsymbol{X}_k) \geq \hat{\gamma}_t\}}\, S(\boldsymbol{X}_i)}, \tag{52}$$

$i = 2, \ldots, n$.

However, other numerical experiments suggest that high power polynomials, $\psi(s) = s^{\beta}$ with large β, and the Boltzmann function $\psi(s) = \mathrm{e}^{-s/\beta}$ are not advisable, as they may lead more easily to local minima.

4.2. *Fully adaptive CE algorithm*

We present a modification of Algorithm 2.3, called the *fully automated CE* (FACE) algorithm in which the sample size is updated adaptively at each iterations t of the algorithm, i.e., $N = N_t$. In addition, this modification is able to identify some "difficult" and pathological problems.

Consider a *maximization* problem and let

$$S_{t,(1)} \leq \cdots \leq S_{t,(N_t)}$$

denote the *ordered* sample performances of the N_t samples at the t-th iterations. To ease notations, we denote $S_{t,(N_t)}$ by S_t^*.

The main assumption in the FACE Algorithm is that at the end of each iteration t the updating of the parameters is done on the basis of a *fixed* number, N^{elite} say, of the best performing samples, the so-called *elite* samples. Thus, the set of elite samples $\mathcal{E}_t$ consist of those N^{elite} samples in $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{N_t}\}$ for which the performances $S(\boldsymbol{X}_1), \ldots, S(\boldsymbol{X}_{N_t})$ are highest. The updating steps 2 and 3 of Algorithm 2.3 are modified such that

$$\hat{\gamma}_t = S_{(N_t - N^{\text{elite}} + 1)},$$

and

$$\hat{\mathbf{v}}_t = \operatorname*{argmax}_{\mathbf{v}} \sum_{\boldsymbol{X}_i \in \mathcal{E}_t} \ln f(\boldsymbol{X}_i; \mathbf{v}).$$

Note that $\hat{\gamma}_t$ is equal to the worst sample performance of the best N^{elite} sample performances, and S_t^* is the best of the elite performances (indeed, of all performances).

In the FACE algorithm the parameters ρ and N of Algorithm 2.3 are updated adaptively. Specifically, they are "replaced" by a single parameter: the number of elite samples N^{elite}. The above updating rules are consistent with Algorithm 2.3 provided we view ρ in Algorithm 2.3 as the parameter which changes inversely proportional to N_t: $\rho_t = N^{\text{elite}}/N_t$.

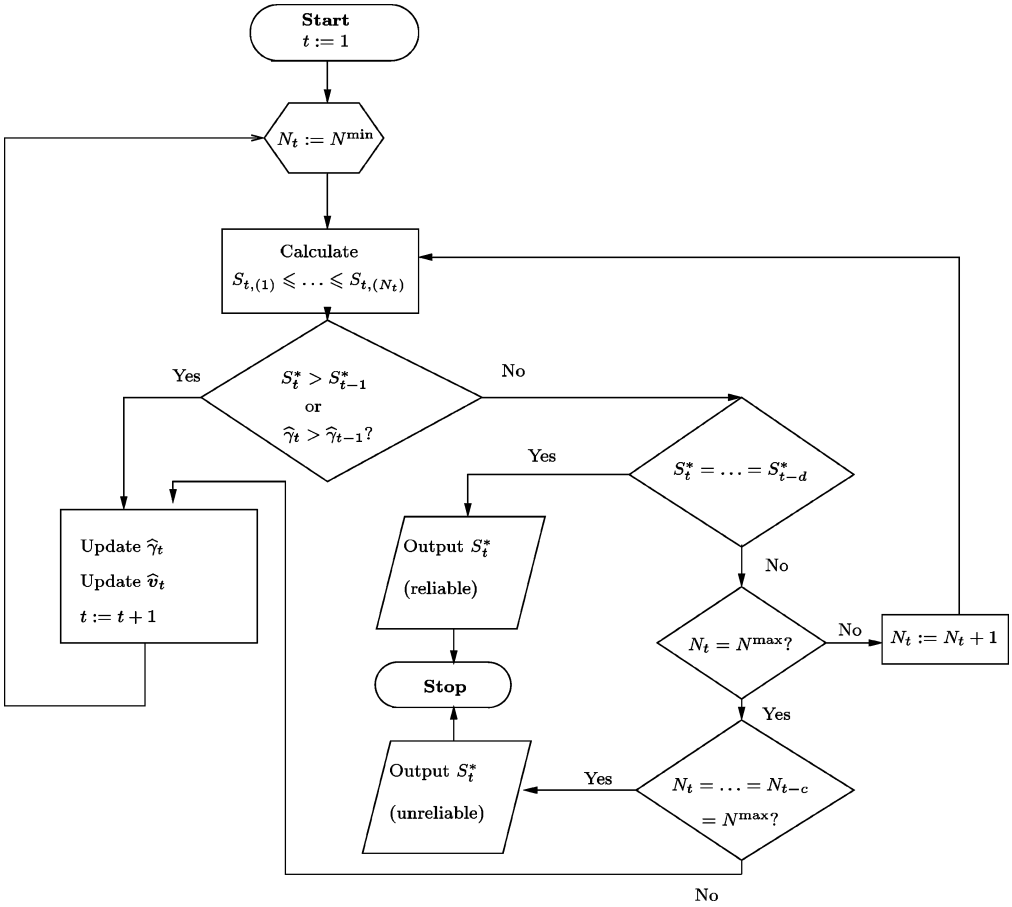

Figure 8. The flowchart for the FACE algorithm.

It was found experimentally that a sound choice is $N^{\text{elite}} = c_0\, n$ and $N^{\text{elite}} = c_0\, n^2$ for SNNs and SENs, respectively, where c_0 is a fixed positive constant (usually in the interval $0.01 \leq c_0 \leq 0.1$). The easiest way to explain FACE is via the flow chart in figure 8.

For each iteration t of FACE algorithm we design a sampling plan which ensures with high probability that

$$S_t^* > S_{t-1}^*. \tag{53}$$

Note that (53) implies *improvement* of the maximal order statistics (best elite performance) at each iteration. To ensure (53) with high probability, we allow N_t to vary at

each iteration t in a quite wide range

$$N^{\min} \leq N_t \leq N^{\max},$$

where, say, for an SNN-type problem, $N^{\min} = n$ and $N^{\max} = 20\,n$. Note that we always start each iteration by generating $N^{\min}$ samples. If at any iteration t, while increasing N_t we obtain simultaneously that $N_t = N^{\max}$ and (53) is violated, then we directly proceed with updating $(\hat{\gamma}_t, \hat{\mathbf{v}}_t)$ according to Algorithm 2.3 before proceeding with the next iteration of the FACE Algorithm. However, if FACE keeps generating samples of size $N^{\max}$ for several iterations in turn, say, for $c = 3$ iterations and (53) is violated then we stop, and announce that FACE identified a "hard" problem for which the estimate of the optimal solution is *unreliable*.

Besides if $S_{t,(N_t)} > S_{t-1,(N_{t-1})}$, we also continue with the next iteration if

$$\hat{\gamma}_t > \hat{\gamma}_{t-1}. \tag{54}$$

Note that (54) implies *improvement* of the worst elite sample performance $\hat{\gamma}_t$ at each iteration.

Similar to Algorithm 2.3 we initialize by choosing some $\hat{\mathbf{v}}_0$. For example, for the max-cut problem we choose (using $\boldsymbol{p}$ instead of $\mathbf{v}$) $\hat{\boldsymbol{p}}_0 = \boldsymbol{p}_0 = (1, 1/2, \ldots, 1/2)$. We assume that the FACE parameters $N^{\min}$, $N^{\max}$, α and the stopping constant d are chosen in advance. For example, for the max-cut problem we take $N^{\min} = n$. We let $t = 1$ and $N_1 = N^{\min}$ and proceed as follows:

Algorithm 4.1 (FACE Algorithm)

1. At iteration t, $t = 1, 2, \ldots$ take an initial sample of size N_t, with $N^{\min} \leq N_t \leq N^{\max}$ from $f(\cdot; \hat{\mathbf{v}}_{t-1})$.
2. If (53) or (54) holds, proceed with the updating steps (20) and (22) using the N_t samples in Step 1.
3. If (53) and (54) are violated, check whether or not

$$S_t^* = \cdots = S_{t-d}^*. \tag{55}$$

   If so, stop and deliver S_t^* as an estimate of the optimal solution. Call such S_t^* a **reliable** estimate of the optimal solution. If (55) does not hold, and if $N_t < N^{\max}$, increase N_t by 1, recalculate S_t^* and $\hat{\gamma}_t$, and go to Step 2.
4. If $N_t = N^{\max}$ and each of (53), (54) and (55) is violated, proceed with (20) and (22) using the $N^{\max}$ samples mentioned in Step 1 and go to step 2.
5. If $N_t = N^{\max}$ for several iterations in turn, say for $c = 3$ iterations, and each of (53), (54) and (55) are violated, stop and announce that FACE identified a "hard" problem. Call $S_{t,(N_t)}$ an **unreliable** estimate of the optimal solution.

The stopping criterion (55) means that the best samples in the last d iterations are the same. Note that if (53) holds for *all* $t \geq 1$ we automatically obtain that $N_t = N^{\min}$ for all t. In such case FACE reduces to the original Algorithm 2.3. We found that FACE typically speeds up the convergence up to 2 times as compared to CE with fixed N and ρ.

5. Further developments

In this section we present two additional applications of the CE method. In Section 5.1 we consider the problem of vector quantization and clustering. We describe a straightforward application of the CE method to this problem, and demonstrate high performance of the CE method. In Section 5.2 we consider the problem of learning the optimal policy in Markovian decision processes.

5.1. Vector quantization and clustering analysis

The clustering problem reads as follows: given a dataset $\mathcal{Z} = \{z_1, \ldots, z_n\}$ of points in some d-dimensional Euclidean region, partition the data into K "clusters" $R_1, \ldots, R_K$ (with $R_i \cap R_j = \emptyset$, for $i \neq j$, and $\cup_j R_j = \mathcal{Z}$), such that some empirical *loss function* is minimized. A typical loss function is (Webb, 1999):

$$\sum_{j=1}^{K} \sum_{z \in R_j} \|z - \boldsymbol{c}_j\|^2, \tag{56}$$

where

$$\boldsymbol{c}_j = \frac{1}{|R_j|} \sum_{z \in R_j} z \tag{57}$$

presents the *centroid* of the points of cluster R_j. Denoting by $\boldsymbol{x} = (x_1, \ldots, x_n)$ the vector with $x_i = j$ when $z_i \in R_j$, and letting $z_{ij} = I_{\{x_i = j\}} z_i$, we can write (56) as the following function of $\boldsymbol{x}$:

$$S(\boldsymbol{x}) = \sum_{j=1}^{K} \sum_{i=1}^{n} I_{\{x_i = j\}} \|z_{ij} - \boldsymbol{c}_j\|^2, \tag{58}$$

where the centroids are given as

$$\boldsymbol{c}_j = \frac{1}{n_j} \sum_{i=1}^{n} z_{ij},$$

with $n_j = \sum_{i=1}^{n} I_{\{x_i = j\}}$ the number of points in the jth cluster.

As we mentioned, our goal is to find a partition $\{R_j\}$ and a corresponding vector of centroids $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K)$ that minimizes (56). In the terminology of *vector quantization*

(Webb, 1999), we wish to "quantize" or "encode" the vectors in $\mathcal{Z}$ in such a way that each vector is represented by one of K *source vectors* $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K$, such that the loss (56) of this representation is minimized. Most well-known clustering and vector quantization methods update the vector of centroids, starting from some initial choice $(\boldsymbol{c}_{0,1}, \ldots, \boldsymbol{c}_{0,K})$ and using iterative (typically gradient-based procedures). It is important to realize that in that case (56) is seen as a function of the *centroids*, where each point $\boldsymbol{z}$ is assigned to the nearest centroid, thus determining the clusters. It is well known these type of problems (optimization with respect to the centroids) are multi-extremal and depending on the initial value of the clusters the gradient-based procedures converge to a *local minimum* rather than global minimum. A standard heuristic to minimize (57) is the *K-means algorithm* (Webb, 1999).

We can optimize (58) via the CE method, by viewing it as a minimal cut (min-cut) problem with K partitions. In particular, each partition $R_1, \ldots, R_K$ is represented via a partition vector $\boldsymbol{x} \in \mathcal{X} = \{1, \ldots, n\}^K$ as defined above. The "trajectory generation" of the CE Algorithm 2.3 consists of drawing random $\boldsymbol{X} \in \mathcal{X}$ according to an n-dimensional discrete distribution with independent marginals, such that $\mathbb{P}(X_i = j) = p_{ij}$, $i = 1, \ldots, m$, $j = 1, \ldots, K$. The updating rules are again very simple. For $K = 2$ we may, alternatively, use $\boldsymbol{X} \sim \mathsf{Ber}(\boldsymbol{p})$. In that case the updating rules are given in (39), with $\geq$ replaced with $\leq$.

Example 5.1. Let $n = 5$ and $K = 2$. To generate a feasible cluster we draw $\boldsymbol{X}$ from a 5-dimensional Bernoulli distribution $\mathsf{Ber}(\boldsymbol{p})$ with independent marginals. Assume that the outcome from a particular generation from $\mathsf{Ber}(\boldsymbol{p})$ is $\boldsymbol{x} = (1, 0, 0, 1, 0)$. The associated partition is, clearly, $\{R_1, R_2\} = \{\{1, 4\}, \{2, 3, 5\}\}$. In this case the loss and the centroids can be explicitly calculated, provided the points z_1, z_2, z_3, z_4, z_5 are given. Namely, the loss is

$$\begin{aligned}&(\|z_1 - \boldsymbol{c}_1\|^2 + \|z_4 - \boldsymbol{c}_1\|^2)\\ &+ (\|z_2 - \boldsymbol{c}_2\|^2 + \|z_3 - \boldsymbol{c}_2\|^2 + \|z_5 - \boldsymbol{c}_2\|^2),\end{aligned} \tag{59}$$

and the centroids are

$$\boldsymbol{c}_1 = \frac{1}{2}(z_1 + z_4), \ \boldsymbol{c}_2 = \frac{1}{3}(z_2 + z_3 + z_5), \tag{60}$$

respectively.

The CE method for clustering works well even if the dataset is noisy. This is in contrast to most clustering methods that often require stochastic approximation procedures, which are very slow.

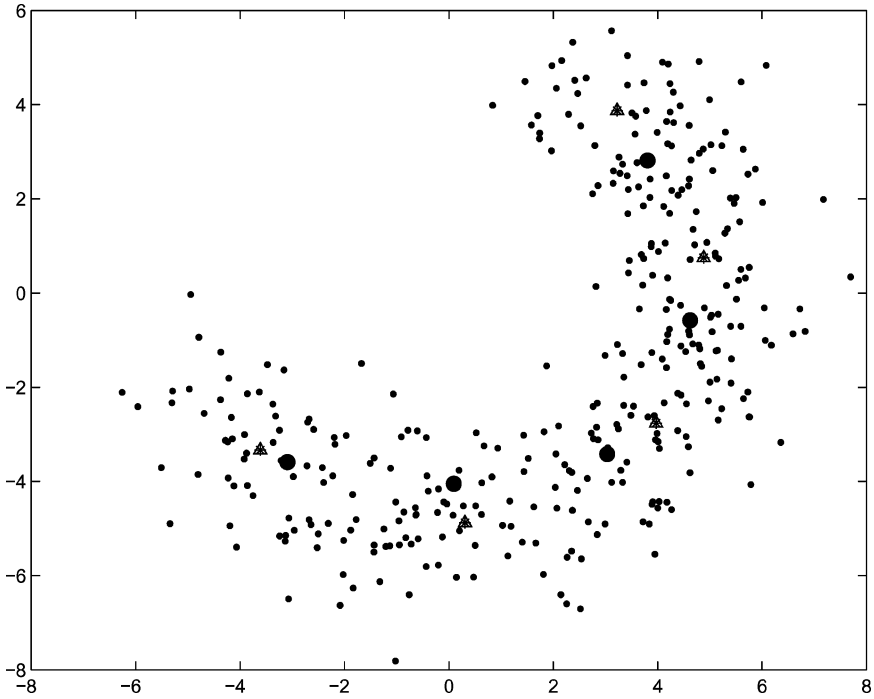

Figure 9. The CE results for vector quantization of the 2-D *banana* data set. Circles designate the final cluster centers produced by CE. Triangles are cluster centers of the K-means algorithm.

Numerical example

Consider as an application of CE the minimization of the loss function, given by (56) for a simple 2-dimensional clustering data set with $m = 5$ clusters and 700 points (depicted by the small dots in figure 9). The resulting locations of the cluster centers obtained by the CE Algorithm are depicted in figure 9 by the large filled circles. Figure 9 also compares the performance of the CE method with the standard K-means one (circles and triangles in figure 9 correspond to the final cluster centers produced by CE and the K-means algorithm, respectively). Although we found that K-means is a little faster (it takes approximately 2 seconds for K-means versus 49 for CE), it readily follows from figure 9 that CE is more accurate than K-means, (the average distortion is 1.428 and 2.408 for CE and K-means, respectively).

5.2. Markovian decision process

The *Markovian decision process* (MDP) model is standard in artificial intelligence, machine learning, operation research and related fields. We review briefly some basic

definitions and concepts related to the MDP model. For details see for example Puterman (1994) and Bertsekas (1995).

An MDP is defined by a tuple $(\mathcal{Z}, \mathcal{A}, \mathcal{P}, r)$ where

1. $\mathcal{Z} = \{1, \ldots, n\}$ is a finite set of states.
2. $\mathcal{A} = \{1, \ldots, m\}$ is the set of possible actions by the decision maker. We assume it is the same for every state – to ease notations.
3. $\mathcal{P}$ is the transition probability matrix with elements $\mathcal{P}(z' \mid z, a)$ presenting the transition probability from state z to state z', when action a is chosen.
4. $r(z, a)$ is the reward for performing action a in state z (r may be a random variable).

At each time instance k the decision maker observes the current state z_k, and determines the action to be taken (say a_k). As a result, a reward given by $r(z_k, a_k)$, denoted by r_k, is received and a new state z' is chosen according to $\mathcal{P}(z' \mid z_k, a_k)$. A *policy* π is a rule that determines, for each history $H_k = z_1, a_1, \ldots, z_{k-1}, a_{k-1}, z_k$ of states and actions, the probability distribution of the decision maker's actions at time k. A policy is called *Markov* if each action depends only on the current state z_k and time k. A Markov policy is called *stationary* if it does not depend on the time k. The goal of the decision maker is to maximize a certain reward as a function of the policy. There are several reward criteria of interest, see Puterman (1994) for a discussion. In this exposition we consider the *total reward* criterion which applies when there exists a finite stopping time τ at which the process terminates. The objective is to maximize the total reward

$$V(\pi, z_0) = \mathbb{E}_\pi \sum_{k=0}^{\tau-1} r_k, \tag{61}$$

starting from some fixed state z_0. Here $\mathbb{E}_\pi$ denotes the expectation with respect to some probability measure induced by the policy π. We will restrict our attention to *stochastic shortest path* MDPs, where it is assumed that the process starts from a specific initial state $z_0 = z^{\text{start}}$, and that there is a absorbing state z^{fin} with zero reward. The objective is given in (61), with τ being the stopping time at which z^{fin} is reached (which we will assume will always happen eventually). For stochastic shortest path MDPs, it is a well known (Puterman, 1994) fact that there exists a stationary Markov policy which maximizes $V(\pi, z_0)$.

If the model (r and $\mathcal{P}$) is known, then there are several efficient methods, such as *value iteration* and *policy iteration*, for finding the optimal policy (Puterman, 1994). However, when the transition probability or the reward in an MDP are *unknown*, the problem is much more difficult, and is referred to as a *learning* one. A well-known framework for learning algorithms is *reinforcement learning* (RL), where an agent learns the behavior of the system through trial-and-error with an unknown dynamic environment, see Kaelbling, Littman, and Moore (1996). For reviews of RL see Barto and Sutton

(1998), Bertsekas and Tsitsiklis (1995), Kaelbling, Littman, and Moore (1996). Note that many of learning algorithm for MDPs rely on the Stochastic Approximation Algorithm and convergence is typically slow. Using the CE method to search for the optimal policy is similar to certain approaches in RL that attempt to search the policy space (or a subset thereof). Notable policy search algorithms used in RL are: a direct search in the policy space of Rosenstein and Barto (2001); a policy gradient search approach of Baxter, Bartlett, and Weaver (2001); and the actor critic framework of Barto, Sutton, and Anderson (1983). See also Sutton et al. (2000), and Konda and Tsitsiklis (2003).

Policy learning via the CE method

Since for the shortest path MDP an optimal stationary policy exists, we can represent each stationary policy as a vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ with $x_i \in \{1, \ldots, m\}$ being the action taken when visiting state i. Writing the expectation in 61 as

$$S(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{x}} \sum_{k=0}^{\tau-1} r(Z_k, A_k), \tag{62}$$

where $Z_0, Z_1, \ldots$ are the states visited, and $A_0, A_1, \ldots$ the actions taken, we see that the optimization problem 61 is of the form 30. From a CE point of view we consider the maximization problem 62 as a *noisy* maximization problem. The idea now is to combine the random policy generation and the random trajectory generation in the following way: At each stage of the CE algorithm we generate random policies and random trajectories using an auxiliary $n \times m$ matrix $P = (p_{za})$, such that for each state z we choose action a with probability p_{za}. Once this "policy matrix" P is defined, each iteration of the CE algorithm comprises the following two standard phases:

1. Generation of N random trajectories $(Z_0, A_0, Z_1, A_1, \ldots, Z_\tau, A_\tau)$ using the auxiliary policy matrix P. The cost of each trajectory is computed via

$$\hat{S}(\boldsymbol{X}) = \sum_{j=0}^{\tau-1} r(Z_k, A_k). \tag{63}$$

2. Updating of the parameters of the policy matrix (p_{za}) on the basis of the data collected at the first phase.

The matrix P is typically initialized to a uniform matrix $(p_{ij} = 1/m)$. Generation of random trajectories for MDP is straightforward and is given for completeness. All one has to do is to start the trajectory from the initial state $z_0 = z^{\text{start}}$ and follow the trajectory by generating at each new state according to the probability distribution of P, until the absorbing state z^{fin} is reached at time τ, say.

Algorithm 5.1 (Trajectory generation for MDP)
Input: P auxiliary policy matrix.

1. Start from the given initial state $Z_0 = z^{\text{start}}$, set $k = 0$.
2. Generate an action A_k according to the Z_kth row of P, calculate the reward $r_k = r(Z_k, A_k)$ and generate a new state Z_{k+1} according to $\mathcal{P}(\cdot \mid Z_k, A_k)$. Set $k = k + 1$. Repeat until $z_k = z^{\text{fin}}$.
3. Output the total reward of the trajectory $(Z_0, A_0, Z_1, A_1, \ldots, Z_\tau)$, given by (63).

Given the N trajectories $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ and their scores, $\hat{S}(\boldsymbol{X}_1), \ldots, \hat{S}(\boldsymbol{X}_N)$, one can update the parameter matrix (p_{za}) using the CE method, namely as per

$$\hat{p}_{t,za} = \frac{\sum_{k=1}^{N} I_{\{\hat{S}(\boldsymbol{X}_k) \geq \hat{\gamma}_t\}} I_{\{\boldsymbol{X}_k \in \mathcal{X}_{za}\}}}{\sum_{k=1}^{N} I_{\{\hat{S}(\boldsymbol{X}_k) \geq \hat{\gamma}_t\}} I_{\{\boldsymbol{X}_k \in \mathcal{X}_z\}}}, \tag{64}$$

where the event $\{\boldsymbol{X}_k \in \mathcal{X}_z\}$ means that the trajectory $\boldsymbol{X}_k$ contains a visit to state z and the event $\{\boldsymbol{X}_k \in \mathcal{X}_{za}\}$ means the trajectory corresponding to policy $\boldsymbol{X}_k$ contains a visit to state z in which action a was taken.

We now explain how to take advantage of the Markovian nature of the problem. Let us think of a maze where a certain trajectory starts badly, that is, the path is not efficient in the beginning, but after some time it starts moving quickly towards the goal. According to (64), all the updates are performed in a similar manner in every state in the trajectory. However, the actions taken in the states that were sampled near the target were successful, so one would like to "encourage" these actions. Using the Markovian property one can substantially improve the above algorithm by considering for each state the part of the reward from the visit to that state onwards. We therefore use the same trajectory and simultaneously calculate the performance for every state in the trajectory separately. The idea is that each choice of action in a given state affects the reward from that point on, disregarding the past.

The sampling Algorithm 5.1 does not change in Steps 1 and 2. The difference is in Step 3. Given a policy $\boldsymbol{X}$ and a trajectory $(Z_0, A_0, Z_1, A_1, \ldots, Z_\tau, A_\tau)$ we calculate the performance from every state until termination. For every state $z = Z_j$ in the trajectory the (estimated) performance is $\hat{S}_z(\boldsymbol{X}) = \sum_{k=j}^{\tau-1} r_k$. The updating formula for $\hat{p}_{za}$ is similar to (64), however *each state z is updated separately* according to the (estimated) performance $\hat{S}_z(\boldsymbol{X})$ obtained from state z onwards.

$$\hat{p}_{t,za} = \frac{\sum_{k=1}^{N} I_{\{\hat{S}_z(\boldsymbol{X}_k) \geq \hat{\gamma}_{t,z}\}} I_{\{\boldsymbol{X}_k \in \mathcal{X}_{za}\}}}{\sum_{k=1}^{N} I_{\{\hat{S}_z(\boldsymbol{X}_k) \geq \hat{\gamma}_{t,z}\}} I_{\{\boldsymbol{X}_k \in \mathcal{X}_z\}}}. \tag{65}$$

A crucial point here is to understand that in contrast to (64) the CE optimization is carried for every state separately and a different threshold parameter $\hat{\gamma}_{t,z}$ is used for every state z, at iteration t. This facilitates faster convergence for "easy" states where the optimal

policy is easy to find. Numerical results indicate that the CE algorithm with updating (65) is much faster then that with updating (64).

Numerical results

The CE method with the updating rule (65) and trajectory generation according to Algorithm 5.1 was implemented for a maze problem, which presents a two-dimensional grid world. We assume that

1. The moves in the grid are allowed in four possible directions with the goal to move from the upper-left corner to the lower-right corner.
2. The maze contains obstacles ("walls") into which movement is not allowed.
3. The reward for every allowed movement until reaching the goal is -1.

In addition we introduce:

1. A small (failure) probability not to succeed moving in an allowed direction.
2. A small probability of succeeding moving in the forbidden direction ("moving through the wall").
3. A high cost for the moves in a forbidden direction.

In figure 10 we present the results for 20×20 maze. We set the following parameters: $N = 1000$, $\rho = 0.03$, $\alpha = 0.7$. The initial policy was a uniformly random one. The cost of the moves were assumed to be random variables uniformly distributed between 0.5 and 1.5 and uniformly distribute between 25 and 75 for the allowed and forbidden moves, respectively. Note that the expected cost for the allowed and forbidden moves are equal to 1 and 50, respectively. The success probabilities in the allowed and forbidden states were taken 0.95 and 0.05, respectively. The arrows $z \to z'$ in figure 10 indicate that at the current iteration the probability of going from z to z' is at least 0.01. In other words, if a corresponds to the action that will lead to state z' from z then we will plot an arrow from z to z' provided $p_{za} > 0.01$.

In all our experiments CE found the target exactly, within 5–10 iterations and CPU time was less than one minute (on a 500 MHz Pentium processor). Note that the successive iterations of the policy in figure 10 quickly converge to the optimal policy.

Additional numerical results can be found in Dubin (2002). Extensions, other reward criteria, and further applications appear in Mannor, Rubinstein, and Gat (2003). These results demonstrate that the CE algorithm may serve as a policy learning algorithm. In contrast to the popular Q-learning (e.g. Dayan and Watkins, 1992) and value iteration (e.g. Bertsekas and Tsitsiklis, 1995) based algorithms, no information is propagated back, but rather the policy is learned directly.

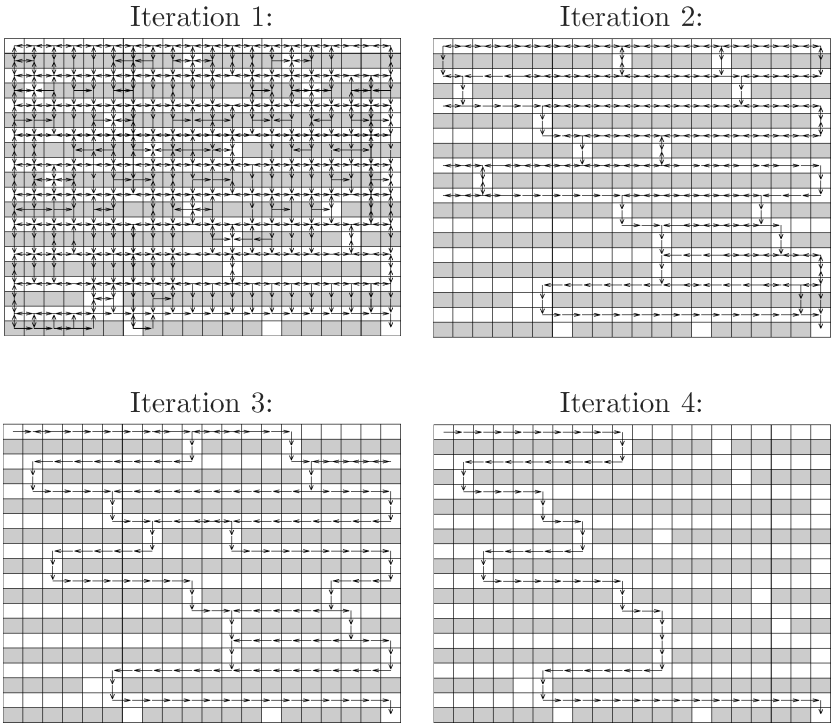

Figure 10. Performance of the CE Algorithm for the 20×20 maze. Each arrow indicates a probability > 0.01 of going in that direction.

6. Discussion and future directions

In this tutorial we introduced the CE method, a new generic approach to efficiently solve difficult deterministic and stochastic optimization problems (both combinatorial and continuous multi-extremal), and to adaptively find efficient estimators for rare event probabilities. We presented the CE methodology, the basic algorithm and its modifications, and discussed its applications to several combinatorial optimization and machine learning problems. We showed that the CE method is a simple, versatile and easy usable tool. Another virtue of the CE method is its robustness with respect to its parameters. In most cases, there is a wide range of parameters N, ρ, α and $\boldsymbol{p}$, P that leads to the desired solution with high probability, although sometimes choosing an appropriate parametrization and sampling scheme is more of an art than a science. Moreover, the foundations of the CE method are based on some well known classical probabilistic and statistical principles, and are therefore fairly problem-independent.

Although, as mentioned, it is out of the scope of this tutorial to give a detailed comparison between the CE method and other heuristics we shall point out a few similarities and differences. The main difference between the CE method and simulating annealing

(Aarts and Korst, 1989) is that the latter can be viewed as a local search algorithm whereas the CE is a global search one, and thus, it is less likely – at least in principle – to get stuck in a local optimal solution. The CE method and genetic algorithms (Goldberg, 1989) share the common idea of picking samples at random and improve the way the samples are generated from generation to generation. In both methods, the solutions fitness is estimated according to a score/performance function. When running genetic algorithms the genetic encoding of the solution is problem-specific and often requires significant effort. In addition, there are many parameters that appear to be critical - crossover and mutation probabilities and sample size. The way the CE method chooses the next sample is less ambiguous, as the change of population generation parameter is optimal in "Kullback-Leiber distance" sense. Also the ant colony optimization (ACO) method (Dorigo, Di Caro, and Gambardella, 1999) bears similarities with the CE method, especially for SEN. The difference is the way the next generation of solutions is generated. In ACO good solutions in the previous generation increase the probability that solutions in subsequent generations follow a similar path. The generation of future solutions is not based on a principled calculation, but rather on problem dependent heuristics. We note that methods in the spirit of tabu search (Glover and Laguna, 1993) and guided local search (Voudouris, 2003) that penalize the neighborhood of previously examined solutions, use a completely different mechanism. Finally, we do not claim that the CE method always finds the optimal solution. Although the numerical results with CE are typically quite reliable, there is no guarantee that it will not get trapped in a local minimum, at times.

There are several research topics that call for further development. First, the resilience of the CE method to local minima has been observed in many applications but is still not well understood. It seems that the CE method tends to overcome local minima based on both the randomization mechanism involving the elite sampling and the smoothed updating.

Second, the convergence of the CE method deserves further study. Convergence results concerning the CE method can be found in several sources. Typically, these are asymptotic in nature, similar to existing convergence results for genetic algorithms, simulated annealing and ant colony algorithms. As a result they do not always shed much light on the actual (non-asymptotic) behavior of the algorithm. In the context of rare event simulation, which can be readily adopted to combinatorial optimization, convergence of CE is given in Homem-de-Mello and Rubinstein (2002). The basic assumption there is that the probability of a rare event does not vanish in a neighborhood of the optimal solution. The algorithm proposed in Homem-de-Mello and Rubinstein (2002) is the same as Algorithm 2.1, only that ρ and N are determined adaptively. Another convergence theorem in the rare event setting is given in Lieber (1998). Margolin (2005) provides asymptotical convergence results of two modifications of the main CE Algorithm 2.3. The proof is similar to that of Gutjahr (2000) for the ant colony optimization. This asymptotic result provides no guidance on convergence rates. Much more on the convergence of CE should be done, in particular finding bounds of convergence rates, at least for some particular difficult optimization problems.

Another research issue is to develop and investigate some alternative stopping rules. The current heuristic rule only indicates the lack of improvement in the past few iterations and when the CE method stops it may happen that it is not even a local optimum. Finally, we consider application of the CE method to constrained optimization problems, and in particular to constrained integer programs, as a challenging topic which would be worth exploring.

For further reading we refer to the monograph (Rubinstein and Kroese, 2004).

Acknowledgments

We would like to thank the associate editor and the two anonymous referees for their detailed and most valuable comments. The research of R.Y. Rubinstein was supported by the Israel Science Foundation (grant No. 191-565).

References

Aarts, E.H.L. and J.H.M Korst. (1989). *Simulated Annealing and Boltzmann Machines*. John Wiley & Sons.

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* 134, 19–67.

Asmussen, S., D.P. Kroese, and R.Y. Rubinstein. (2005). "Heavy Tails, Importance Sampling and Cross-Entropy." *Stochastic Models* 21(1). To appear.

Barto, A., R. Sutton, and C. Anderson. (1983). "Neuron-Like Adaptive Elements that can Solve Difficult Learning Control Problems." *IEEE Transactions on Systems, Man, and Cybernetics* 13, 834–846.

Barto, A.G. and R.S. Sutton. (1998). *Reinforcement Learning*. MIT Press.

Baxter, J., P.L. Bartlett, and L. Weaver. (2001). Experiments with Infinite-Horizon, Policy-Gradient Estimation." *Journal of Artificial Intelligence Research* 15, 351–381.

Bertsekas, D.P. (1995). *Dynamic Programming and Optimal Control*. Athena Scientific.

Bertsekas, D.P. and J.N. Tsitsiklis. (1995). *Neuro-Dynamic Programming*. Athena Scientific.

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* 134, 153–181.

Cohen, I., B. Golany, and A. Shtub. (2005). "Managing Stochastic Finite Capacity Multi-Project Systems Through the Cross-Entropy Method." *Annals of Operations Research* 134, 183–199.

Colorni, A., M. Dorigo, F. Maffioli, V. Maniezzo, G. Righini, and M. Trubian. (1996). "Heuristics from Nature for Hard Combinatorial Problems." *International Transactions in Operational Research* 3(1), 1—21.

Dayan, P. and C. Watkins (1992). "Q-Learning." *Machine Learning* 8, 279–292.

de Boer, P.T. (2000). "Analysis and Efficient Simulation of Queueing Models of Telecommunication Systems." Ph.D. thesis, University of Twente.

de Boer, P.T., D.P. Kroese, and R.Y. Rubinstein. (2002). "Estimating Buffer Overflows in Three Stages using Cross-Entropy." In *Proceedings of the 2002 Winter Simulation Conference*. San Diego, pp. 301–309.

de Boer, P.T., D.P. Kroese, and R.Y. Rubinstein. (2004). "A Fast Cross-Entropy Method for Estimating Buffer Overflows in Queueing Networks." *Management Science* 50(7), 883–895.

Dorigo, M., G. Di Caro, and L.M. Gambardella. (1999). "Ant Algorithms for Discrete Optimization." *Artificial Life* 5(2), 137–172.

Dubin, U. (2002). "Application of the Cross-Entropy Method to Neural Computation." Master's thesis, Technion, Electrical Engineering.

Dubin, U. (2004). "Application of the Cross-Entropy Method for Image Segmentation." Unpublished.

Garey, M.R. and D.S. Johnson. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: W.H. Freeman and Company.

Glover, F. and M. Laguna. (1993). *Modern Heuristic Techniques for Combinatorial Optimization*, chapter 3: Tabu search. Blackwell Scientific Publications.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley.

Gutjahr, W.J. (2000). "A Graph-Based Ant System and Its Convergence." *Future Generations Computing* 16, 873–888.

Helvik, B.E. and O. Wittner. (2001). "Using the Cross-Entropy Method to Guide/Govern Mobile Agent's Path Finding in Networks." In *3rd International Workshop on Mobile Agents for Telecommunication Applications – MATA'01*.

Homem-de-Mello, T. and R.Y. Rubinstein. (2002). "Rare Event Estimation for Static Models via Cross-Entropy and Importance Sampling." (submitted for publication).

Hui, K.-P., N. Bean, M. Kraetzl, and D.P. Kroese. (2005). "The Cross-Entropy Method for Network Reliability Estimation." *Annals of Operations Research* 134, 101–118.

Kaelbling, L.P., M. Littman, and A.W. Moore. (1996). "Reinforcement Learning – A Survey." *Journal of Artificial Intelligence Research* 4, 237–285.

Keith, J. and D.P. Kroese. (2002). "SABRES: Sequence Alignment By Rare Event Simulation." In *Proceedings of the 2002 Winter Simulation Conference*, San Diego, pp. 320–327.

Konda, V.R. and J.N. Tsitsiklis. (2003). "Actor-Critic Algorithms." SIAM *Journal on Control and Optimization* 42(4), 1134–1166.

Kroese, D.P. and R.Y. Rubinstein. (2004). "The Transform Likelihood Ratio Method for Rare Event Simulation with Heavy Tails." *Queueing Systems* 46, 317–351.

Lieber, D. (1998). "Rare-Events Estimation via Cross-Entropy and Importance Sampling." Ph.D. thesis, William Davidson Faculty of Industrial Engineering and Management, Technion, Haifa, Israel.

Mannor, S., R.Y. Rubinstein, and Y. Gat. (2003). "The Cross-Entropy Method for Fast Policy Search." In *Proceedings of the Twentieth International Conference on Machine Learning*. Morgan Kaufmann, pp. 512–519.

Margolin, L. (2002). "Application of the Cross-Entropy Method to Scheduling Problems." Master's thesis, Technion, Industrial Engineering.

Margolin, L. (2004). "The Cross-Entropy Method for the Single Machine Total Weighted Tardiness Problem." Unpublished.

Margolin, L. (2005). "On the Convergence of the Cross-Entropy Method." *Annals of Operations Research* 134, 201–214.

Menache, I., S. Mannor, and N. Shimkin. (2005). "Basis Function Adaption in Temporal Difference Reinforcement Learning." *Annals of Operations Research* 134, 215–238.

Papadimitriou, C.H. and M. Yannakakis. (1991). "Optimization, Approximation, and Complexity Classes." *J. Comput. System Sci.* 43, 425–440.

Puterman, M. (1994). *Markov Decision Processes*. Wiley-Interscience.

Ridder, A. (2005). "Importance Sampling Simulations of Markovian Reliability Systems Using Cross-Entropy." *Annals of Operations Research* 134, 119–136.

Rosenstein, M.T. and A.G. Barto. (2001). "Robot Weightlifting by Direct Policy Search." In B. Nebel (ed.). *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, pp. 839–846.

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

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

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

Rubinstein, R.Y. (2002). "Cross-Entropy and Rare-Events for Maximal Cut and Bipartition Problems." *ACM Transactions on Modeling and Computer Simulation*, 27–53.

Rubinstein, R.Y. and D.P. Kroese. (2004). *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. Springer-Verlag, New York.

Rubinstein, R.Y. and B. Melamed. (1998). *Modern Simulation and Modeling*. Wiley series in probability and Statistics.

Rubinstein, R.Y. and A. Shapiro. (1993). *Discrete Event Systems: Sensitivity Analysis and Stochastic Optimization via the Score Function Method*. Wiley.

Shi, L. and S. Olafsson. (2000). "Nested Partitioning Method for Global Optimization." *Operations Research* 48(3), 390–407.

Sutton, R.S., D. McAllester, S. Singh, and Y. Mansour. (2000). "Policy Gradient Methods for Reinforcement Learning with Function Approximation." In *Advances in Neural Information Processing Systems 12*. MIT Press, pp. 1057–1063.

Voudouris, C. (2003). "Guided Local Search – An Illustrative Example in Function Optimisation." *BT Technology Journal* 16(3), 46–50.

Webb, A. (1999). *Statistical Pattern Recognition*. Arnold.