

---

## A Tutorial Introduction to the Cross-Entropy Method

### 2.1 Introduction

The aim of this chapter is to provide a gentle and self-contained introduction to the cross-entropy (CE) method. We refer to Section 1.1 for additional background information on the CE method, including many references.

We wish to show that

1. the CE method presents a simple, efficient, and general method for solving a great variety of *estimation and optimization* problems, especially NP-hard combinatorial deterministic and stochastic (noisy) problems,
2. the CE method is a valuable tool for *Monte-Carlo simulation*, in particular when very small probabilities need to be accurately estimated (so-called rare-event simulation).

The CE method has its origins in an adaptive algorithm for rare-event simulation, based on variance minimization [144]. This procedure was soon modified [145] to a randomized optimization technique, where the original variance minimization program was replaced by an associated cross-entropy minimization problem; see Section 1.1.

In the field of rare-event simulation, the CE method is used in conjunction with *importance sampling* (IS), a well-known variance reduction technique in which the system is simulated under a different set of parameters, called the *reference 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 conventional IS technique is that the optimal reference parameters to be used in IS are usually very difficult to obtain. Traditional techniques for estimating the optimal reference parameters [148] typically involve time-consuming *variance minimization* (VM) programs. The advantage of the CE method is that it provides a simple and fast adaptive procedure for estimating the optimal reference parameters in the IS. Moreover, the CE method also enjoys asymptotic convergence properties. For example, it is shown in [85] that

for *static* models — cf. Remark 2.3 — 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; see [46, 15] and Chapter 3.

In the field of optimization problems — combinatorial or otherwise — the CE method can be readily applied by first translating the underlying optimization problem into an associated estimation problem, the so-called *associated stochastic problem* (ASP), which typically involves rare-event estimation. Estimating the rare-event probability and the associated optimal reference parameter for the ASP via the CE method translates effectively back into solving the original optimization problem. Many combinatorial optimization problems (COPs) can be formulated as optimization problems concerning a weighted graph. Depending on the particular problem, the ASP introduces randomness in either

- (a) the *nodes* of the graph, in which case we speak of a *stochastic node network* (SNN), or
- (b) the *edges* of the graph, in which case we speak of a *stochastic edge network* (SEN).

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 (TSP), the quadratic assignment problem, the clique problem, and optimal policy search in Markovian decision problems (MDPs). We should emphasize that the CE method may be 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 [24] and in various simulation-based optimization models [148], such as optimal buffer allocation [9]. Chapter 6 deals with noisy optimization problems, for which the CE method is ideally suited.

Recently it was found that the CE method has a strong connection with the fields of neural computation and reinforcement learning. Here CE has been successfully applied to clustering and vector quantization and several MDPs under uncertainty. Indeed, the CE algorithm can be viewed as a stochastic learning algorithm involving the following two iterative phases:

1. Generation of a sample of random data (trajectories, vectors, etc.) according to a specified random mechanism.
2. Updating the parameters of the random mechanism, typically parameters of pdfs, on the basis of the data, to produce a “better” sample in the next iteration.

The significance of the cross-entropy concept is that it defines a precise mathematical framework for deriving fast and “good” updating/learning rules.

The rest of the chapter is organized as follows. In Section 2.2 we present two toy examples that illustrate the basic methodology behind the CE method. The general theory and algorithms are detailed in Section 2.3, for rare-event simulation, and Section 2.4, for Combinatorial Optimization. Finally, in Section 2.5 we discuss the application of the CE method to the max-cut and the TSP, and provide numerical examples of the performance of the algorithm.

Our intention is not to compare the CE method with other heuristics, but demonstrate its beauty and simplicity and promote CE for further applications to optimization and rare-event simulation. This chapter is based partly on [44].

## 2.2 Methodology: Two Examples

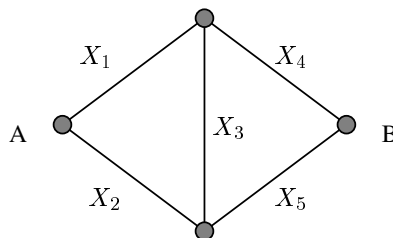
In this section we illustrate the methodology of the CE method via two toy examples, one dealing with rare-event simulation, and the other with combinatorial optimization.

### 2.2.1 A Rare-Event Simulation Example

Consider the weighted graph of Figure 2.1, with random weights  $X_1, \dots, X_5$ . Suppose the weights are independent and exponentially distributed random variables with means  $u_1, \dots, u_5$ , respectively. Denote the probability density function (pdf) of  $\mathbf{X}$  by  $f(\cdot; \mathbf{u})$ ; thus,

$$f(\mathbf{x}; \mathbf{u}) = \exp \left( - \sum_{j=1}^5 \frac{x_j}{u_j} \right) \prod_{j=1}^5 \frac{1}{u_j}. \quad (2.1)$$

Let  $S(\mathbf{X})$  be the total length of the shortest path from node A to node B.



**Fig. 2.1.** Shortest path from A to B.

We wish to estimate from simulation

$$\ell = \mathbb{P}(S(\mathbf{X}) \geq \gamma) = \mathbb{E}I_{\{S(\mathbf{X}) \geq \gamma\}} , \quad (2.2)$$

that is, the probability that the length of the shortest path  $S(\mathbf{X})$  will exceed some fixed  $\gamma$ . A straightforward way to estimate  $\ell$  in (2.2) is to use *crude Monte Carlo* (CMC) simulation. That is, we draw a random sample  $\mathbf{X}_1, \dots, \mathbf{X}_N$  from the distribution of  $\mathbf{X}$  and use

$$\frac{1}{N} \sum_{i=1}^N I_{\{S(\mathbf{x}_i) \geq \gamma\}} \quad (2.3)$$

as the unbiased estimator of  $\ell$ . However, for large  $\gamma$  the probability  $\ell$  will be very small and CMC requires a very large simulation effort. Namely,  $N$  needs to be very large in order to estimate  $\ell$  accurately — that is, to obtain a small relative error of 0.01, say. A better way to perform the simulation is to use *importance sampling* (IS). That is, let  $g$  be another probability density such that  $g(\mathbf{x}) = 0 \Rightarrow I_{\{S(\mathbf{x}) \geq \gamma\}} f(\mathbf{x}) = 0$ . Using the density  $g$  we can represent  $\ell$  as

$$\ell = \int I_{\{S(\mathbf{x}) \geq \gamma\}} \frac{f(\mathbf{x})}{g(\mathbf{x})} g(\mathbf{x}) d\mathbf{x} = \mathbb{E}_g I_{\{S(\mathbf{X}) \geq \gamma\}} \frac{f(\mathbf{X})}{g(\mathbf{X})} , \quad (2.4)$$

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  $\ell$  is

$$\hat{\ell} = \frac{1}{N} \sum_{i=1}^N I_{\{S(\mathbf{x}_i) \geq \gamma\}} W(\mathbf{X}_i) , \quad (2.5)$$

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

$$W(\mathbf{x}) = f(\mathbf{x})/g(\mathbf{x}) \quad (2.6)$$

is called the *likelihood ratio* (LR), and  $\mathbf{X}_1, \dots, \mathbf{X}_N$  is a *random sample* from  $g$ , that is,  $\mathbf{X}_1, \dots, \mathbf{X}_n$  are i.i.d. random vectors with density  $g$ . In the particular case where there is no “change of measure,” that is,  $g = f$ , we have  $W = 1$ , and the LR estimator in (2.6) reduces to the CMC estimator (2.3).

Let us restrict ourselves to  $g$  such that  $X_1, \dots, X_5$  are independent and exponentially distributed with means  $v_1, \dots, v_5$ . Then

$$W(\mathbf{x}; \mathbf{u}, \mathbf{v}) = \frac{f(\mathbf{x}; \mathbf{u})}{f(\mathbf{x}; \mathbf{v})} = \exp \left( - \sum_{j=1}^5 x_j \left( \frac{1}{u_j} - \frac{1}{v_j} \right) \right) \prod_{j=1}^5 \frac{v_j}{u_j} . \quad (2.7)$$

In this case the “change of measure” is determined by the parameter vector  $\mathbf{v} = (v_1, \dots, v_5)$ . The main problem now is how to select a  $\mathbf{v}$  which gives the most accurate estimate of  $\ell$  for a given simulation effort. As we shall see soon 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 next.

**Algorithm**

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

$$\hat{v}_{t,j} = \frac{\sum_{i=1}^N I_{\{S(\mathbf{X}_i) \geq \hat{\gamma}_t\}} W(\mathbf{X}_i; \mathbf{u}, \hat{\mathbf{v}}_{t-1}) X_{ij}}{\sum_{i=1}^N I_{\{S(\mathbf{X}_i) \geq \hat{\gamma}_t\}} W(\mathbf{X}_i; \mathbf{u}, \hat{\mathbf{v}}_{t-1})}. \quad (2.8)$$

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  $\mathbf{X}_1, \dots, \mathbf{X}_{N_1}$  according to the pdf  $f(\cdot; \hat{\mathbf{v}}_T)$  and estimate  $\ell$  via the IS estimator

$$\hat{\ell} = \frac{1}{N_1} \sum_{i=1}^{N_1} I_{\{S(\mathbf{X}_i) \geq \gamma\}} W(\mathbf{X}_i; \mathbf{u}, \hat{\mathbf{v}}_T). \quad (2.9)$$

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. Note also that the algorithm assumes availability of the parameters  $\varrho$  (typically between 0.01 and 0.1),  $N$  and  $N_1$  in advance.

As an example, consider the case where the *nominal* parameter vector  $\mathbf{u}$  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,  $\sqrt{\text{Var}(\hat{\ell})}/\ell$ ) of 0.165. With  $10^8$  samples we got the estimate  $1.30 \cdot 10^{-5}$  with RE 0.03.

Table 2.1 displays the results of the CE method, using  $N = 1000$  and  $\varrho = 0.1$ . This table was computed in less than half a second.

**Table 2.1.** Evolution 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

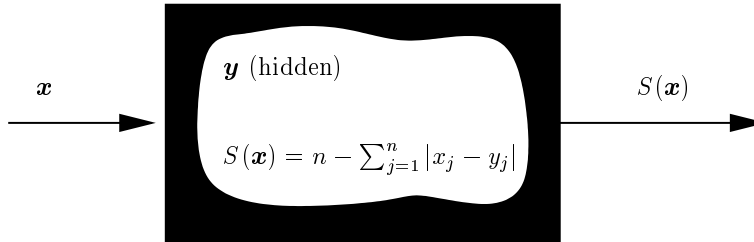
Using the estimated optimal parameter vector of  $\hat{\mathbf{v}}_5 = (1.692, 1.901, 0.129, 0.712, 0.564)$ , the final step with  $N_1 = 10^5$  now gave an estimate of  $1.34 \cdot 10^{-5}$  with an estimated RE of 0.03. The simulation time was only 3 seconds, using a Matlab implementation on a Pentium 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 625.

### 2.2.2 A Combinatorial Optimization Example

Consider a binary vector  $\mathbf{y} = (y_1, \dots, y_n)$ . Suppose that we do not know which components of  $\mathbf{y}$  are 0 and which are 1. However, we have an “oracle” which for each binary *input* vector  $\mathbf{x} = (x_1, \dots, x_n)$  returns the performance or response,

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

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



**Fig. 2.2.** A “device” for reconstructing vector  $\mathbf{y}$ .

A naive way is to repeatedly generate binary vectors  $\mathbf{X} = (X_1, \dots, X_n)$  such that  $X_1, \dots, X_n$  are independent Bernoulli random variables with success probabilities  $p_1, \dots, p_n$ . We write  $\mathbf{X} \sim \text{Ber}(\mathbf{p})$ , where  $\mathbf{p} = (p_1, \dots, p_n)$ . Note that if  $\mathbf{p} = \mathbf{y}$ , which corresponds to the degenerate case of the Bernoulli distribution, we have  $S(\mathbf{X}) = n$ ,  $\mathbf{X} = \mathbf{y}$ , and the naive search algorithm yields the optimal solution with probability 1. The CE method for combinatorial optimization consists of creating a sequence of parameter vectors  $\hat{\mathbf{p}}_0, \hat{\mathbf{p}}_1, \dots$  and

\* Of course, in this toy example the vector  $\mathbf{y}$  can be easily reconstructed from the input vectors  $(0, 0, \dots, 0), (1, 0, \dots, 0), (0, 1, 0, \dots, 0), \dots, (0, \dots, 0, 1)$  only.

levels  $\hat{\gamma}_1, \hat{\gamma}_2, \dots$ , such that  $\hat{\gamma}_1, \hat{\gamma}_2, \dots$ , converges to the optimal performance ( $n$  here) and  $\hat{\mathbf{p}}_0, \hat{\mathbf{p}}_1, \dots$  converges to the optimal degenerated parameter vector that coincides with  $\mathbf{y}$ . Again, the CE procedure — which is similar to the rare-event procedure described in the CE algorithm in Section 2.2.1 — is outlined below, without detail.

### Algorithm

1. Start with some  $\hat{\mathbf{p}}_0$ . Let  $t = 1$ .
2. Draw a sample  $\mathbf{X}_1, \dots, \mathbf{X}_N$  of Bernoulli vectors with success probability vector  $\hat{\mathbf{p}}_{t-1}$ . Calculate the performances  $S(\mathbf{X}_i)$  for all  $i$ , and order them from smallest to biggest,  $S_{(1)} \leq \dots \leq S_{(N)}$ . Let  $\hat{\gamma}_t$  be sample  $(1 - \varrho)$ -quantile of the performances:  $\hat{\gamma}_t = S_{(\lceil (1-\varrho)N \rceil)}$ .
3. Use the **same** sample to calculate  $\hat{\mathbf{p}}_t = (\hat{p}_{t,1}, \dots, \hat{p}_{t,n})$  via

$$\hat{p}_{t,j} = \frac{\sum_{i=1}^N I_{\{S(\mathbf{X}_i) \geq \hat{\gamma}_t\}} I_{\{X_{ij}=1\}}}{\sum_{i=1}^N I_{\{S(\mathbf{X}_i) \geq \hat{\gamma}_t\}}}, \quad (2.10)$$

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

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  $\hat{\mathbf{p}}_t$  has converged to a degenerate — that is, binary — vector. Note that the interpretation of (2.10) is very simple: to *update* the  $j$ -th success probability we count how many vectors of the last sample  $\mathbf{X}_1, \dots, \mathbf{X}_N$  have a performance greater than or equal to  $\hat{\gamma}_t$  and have the  $j$ -th coordinate equal to 1, and we divide this by the number of vectors that have a performance greater than or equal to  $\hat{\gamma}_t$ .

As an example, consider the case where  $\mathbf{y} = (1, 1, 1, 1, 1, 0, 0, 0, 0, 0)$ . Using the initial parameter vector  $\hat{\mathbf{p}}_0 = (1/2, 1/2, \dots, 1/2)$ , and taking  $N = 50$  and  $\varrho = 0.1$ , the algorithm above yields the results given in Table 2.2. We see that the  $\hat{\mathbf{p}}_t$  and  $\hat{\gamma}_t$  converge very quickly to the optimal parameter vector  $\mathbf{p}^* = \mathbf{y}$  and optimal performance  $\gamma^* = n$ , respectively.

**Table 2.2.** Evolution of the sequence  $\{(\hat{\gamma}_t, \hat{\mathbf{p}}_t)\}$ .

$t$	$\hat{\gamma}_t$	$\hat{\mathbf{p}}_t$									
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

*Remark 2.1 (Likelihood ratio term).* Note that the randomized optimization algorithm above is almost the *same* as the rare-event simulation algorithm in the previous Section 2.2.1. The most important difference is the absence of the likelihood ratio term  $W$  in Step 3. The reason is that for the optimization algorithm the choice of the initial parameter  $\mathbf{u}$  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 2.5.

## 2.3 The CE Method for Rare-Event Simulation

In this section 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 2.2.1.

Let  $\mathbf{X} = (X_1, \dots, X_n)$  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  $\mu$ , where  $\mathbf{v}$  is a real-valued parameter (vector). Thus,

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

for any (measurable) function  $H$ . In most (or all) applications  $\mu$  is either a counting measure or the Lebesgue measure. For simplicity, for the rest of this section we take  $\mu(d\mathbf{x}) = d\mathbf{x}$ .

Let  $S$  be some real function on  $\mathcal{X}$ . Suppose we are interested in the probability that  $S(\mathbf{X})$  is greater than or equal to some real number  $\gamma$  — which we will refer to as *level* — under  $f(\cdot; \mathbf{u})$ . This probability can be expressed as

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

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

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

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

is an unbiased estimator of  $\ell$ . However this poses serious problems when  $\{S(\mathbf{X}) \geq \gamma\}$  is a rare event since a large simulation effort is required to estimate  $\ell$  accurately, that is, with a small relative error or a narrow confidence interval.

An alternative is based on importance sampling: take a random sample  $\mathbf{X}_1, \dots, \mathbf{X}_N$  from an *importance sampling* (different) density  $g$  on  $\mathcal{X}$ , and estimate  $\ell$  using the LR estimator (see (2.5))



$$\widehat{\ell} = \frac{1}{N} \sum_{i=1}^N I_{\{S(\mathbf{X}_i) \geq \gamma\}} \frac{f(\mathbf{X}_i; \mathbf{u})}{g(\mathbf{X}_i)} . \quad (2.11)$$

The best way to estimate  $\ell$  is to use the change of measure with density

$$g^*(\mathbf{x}) = \frac{I_{\{S(\mathbf{x}) \geq \gamma\}} f(\mathbf{x}; \mathbf{u})}{\ell} . \quad (2.12)$$

Namely, by using this change of measure we have in (2.11)

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

for all  $i$ ; see [148]. Since  $\ell$  is a constant, the estimator (2.11) 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  $\ell$ . 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 *reference parameter* (sometimes called *tilting parameter*)  $\mathbf{v}$  such that the *distance* between the density  $g^*$  above and  $f(\cdot; \mathbf{v})$  is minimal. A particularly 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:

$$\mathcal{D}(g, h) = \mathbb{E}_g \ln \frac{g(\mathbf{X})}{h(\mathbf{X})} = \int g(\mathbf{x}) \ln g(\mathbf{x}) d\mathbf{x} - \int g(\mathbf{x}) \ln h(\mathbf{x}) d\mathbf{x} . \quad (2.13)$$

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 (2.12) and  $f(\cdot; \mathbf{v})$  is equivalent to choosing  $\mathbf{v}$  such that  $-\int g^*(\mathbf{x}) \ln f(\mathbf{x}; \mathbf{v}) d\mathbf{x}$  is minimized, which is equivalent to solving the maximization problem

$$\max_{\mathbf{v}} \int g^*(\mathbf{x}) \ln f(\mathbf{x}; \mathbf{v}) d\mathbf{x} . \quad (2.14)$$

Substituting  $g^*$  from (2.12) into (2.14) we obtain the maximization program

$$\max_{\mathbf{v}} \int \frac{I_{\{S(\mathbf{x}) \geq \gamma\}} f(\mathbf{x}; \mathbf{u})}{\ell} \ln f(\mathbf{x}; \mathbf{v}) d\mathbf{x} , \quad (2.15)$$

which is equivalent to the program

$$\max_{\mathbf{v}} D(\mathbf{v}) = \max_{\mathbf{v}} \mathbb{E}_{\mathbf{u}} I_{\{S(\mathbf{X}) \geq \gamma\}} \ln f(\mathbf{X}; \mathbf{v}) , \quad (2.16)$$

where  $D$  is implicitly defined above. Again using importance sampling, with a change of measure  $f(\cdot; \mathbf{w})$  we can rewrite (2.16) as

$$\max_{\mathbf{v}} D(\mathbf{v}) = \max_{\mathbf{v}} \mathbb{E}_{\mathbf{w}} I_{\{S(\mathbf{X}) \geq \gamma\}} W(\mathbf{X}; \mathbf{u}, \mathbf{w}) \ln f(\mathbf{X}; \mathbf{v}) , \quad (2.17)$$

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

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

is the *likelihood ratio*, at  $\mathbf{x}$ , between  $f(\cdot; \mathbf{u})$  and  $f(\cdot; \mathbf{w})$ . The optimal solution of (2.17) can be written as

$$\mathbf{v}^* = \operatorname{argmax}_{\mathbf{v}} \mathbb{E}_{\mathbf{w}} I_{\{S(\mathbf{X}) \geq \gamma\}} W(\mathbf{X}; \mathbf{u}, \mathbf{w}) \ln f(\mathbf{X}; \mathbf{v}) . \quad (2.18)$$

We may *estimate*  $\mathbf{v}^*$  by solving the following stochastic program (also called *stochastic counterpart* of (2.17))

$$\max_{\mathbf{v}} \hat{D}(\mathbf{v}) = \max_{\mathbf{v}} \frac{1}{N} \sum_{i=1}^N I_{\{S(\mathbf{X}_i) \geq \gamma\}} W(\mathbf{X}_i; \mathbf{u}, \mathbf{w}) \ln f(\mathbf{X}_i; \mathbf{v}) , \quad (2.19)$$

where  $\mathbf{X}_1, \dots, \mathbf{X}_N$  is a random sample from  $f(\cdot; \mathbf{w})$ . In typical applications the function  $\hat{D}$  in (3.28) is convex and differentiable with respect to  $\mathbf{v}$  [149], in which case the solution of (3.28) may be readily obtained by solving (with respect to  $\mathbf{v}$ ) the following system of equations:

$$\frac{1}{N} \sum_{i=1}^N I_{\{S(\mathbf{X}_i) \geq \gamma\}} W(\mathbf{X}_i; \mathbf{u}, \mathbf{w}) \nabla \ln f(\mathbf{X}_i; \mathbf{v}) = \mathbf{0} , \quad (2.20)$$

The advantage of this approach is that the solution of (2.20) can often be calculated *analytically*. In particular, this happens if the distributions of the random variables belong to a *natural exponential family* (NEF).

It is important to note that the CE program (2.19) is useful only if the probability of the “target event”  $\{S(\mathbf{X}) \geq \gamma\}$  is not too small under  $\mathbf{w}$ , say greater than  $10^{-5}$ . For rare-event probabilities, however, the program (2.19) is difficult to carry out. Namely, due to the rareness of the events  $\{S(\mathbf{X}_i) \geq \gamma\}$ , most of the indicator random variables  $I_{\{S(\mathbf{X}_i) \geq \gamma\}}$ ,  $i = 1, \dots, N$  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 (2.20). A *multilevel* 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  $\gamma_t$  and  $\mathbf{v}_t$  (see Algorithm 2.3.1 below).

We initialize by choosing a not very small  $\varrho$ , say  $\varrho = 10^{-2}$  and by defining  $\mathbf{v}_0 = \mathbf{u}$ . Next, we let  $\gamma_1$  ( $\gamma_1 < \gamma$ ) be such that, under the original density  $f(\mathbf{x}; \mathbf{u})$ , the probability  $\ell_1 = \mathbb{E}_{\mathbf{u}} I_{\{S(\mathbf{X}) \geq \gamma_1\}}$  is at least  $\varrho$ . We then let  $\mathbf{v}_1$  be the optimal CE reference parameter for estimating  $\ell_1$ , 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  $\gamma_t$  is updated, in the second  $\mathbf{v}_t$  is updated. Specifically, starting with  $\mathbf{v}_0 = \mathbf{u}$  we obtain the subsequent  $\gamma_t$  and  $\mathbf{v}_t$  as follows:

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

$$\mathbb{P}_{\mathbf{v}_{t-1}}(S(\mathbf{X}) \geq \gamma_t) \geq \varrho, \quad (2.21)$$

$$\mathbb{P}_{\mathbf{v}_{t-1}}(S(\mathbf{X}) \leq \gamma_t) \geq 1 - \varrho, \quad (2.22)$$

where  $\mathbf{X} \sim f(\cdot; \mathbf{v}_{t-1})$ .

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

$$\hat{\gamma}_t = S_{(\lceil (1-\varrho)N \rceil)}. \quad (2.23)$$

Note that  $S_{(j)}$  is called the  $j$ -th *order-statistic* of the sequence  $S(\mathbf{X}_1), \dots, S(\mathbf{X}_N)$ . Note also that  $\hat{\gamma}_t$  is chosen such that the event  $\{S(\mathbf{X}) \geq \hat{\gamma}_t\}$  is not too rare (it has a probability of around  $\varrho$ ), and therefore updating the reference parameter via a procedure such as (2.23) is not void of meaning.

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

$$\max_{\mathbf{v}} D(\mathbf{v}) = \max_{\mathbf{v}} \mathbb{E}_{\mathbf{v}_{t-1}} I_{\{S(\mathbf{X}) \geq \gamma_t\}} W(\mathbf{X}; \mathbf{u}, \mathbf{v}_{t-1}) \ln f(\mathbf{X}; \mathbf{v}). \quad (2.24)$$

The stochastic counterpart of (2.24) 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

$$\max_{\mathbf{v}} \hat{D}(\mathbf{v}) = \max_{\mathbf{v}} \frac{1}{N} \sum_{i=1}^N I_{\{S(\mathbf{x}_i) \geq \hat{\gamma}_t\}} W(\mathbf{X}_i; \mathbf{u}, \hat{\mathbf{v}}_{t-1}) \ln f(\mathbf{X}_i; \mathbf{v}). \quad (2.25)$$

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

As mentioned before, the optimal solutions of (2.24) and (2.25) can often be obtained *analytically*, in particular when  $f(\mathbf{x}; \mathbf{v})$  belongs to a NEF.

The above rationale results in the following algorithm.

**Algorithm 2.3.1 (Main CE Algorithm for Rare-Event Simulation)**

1. Define  $\hat{\mathbf{v}}_0 = \mathbf{u}$ . Set  $t = 1$  (iteration = level counter).
2. Generate a sample  $\mathbf{X}_1, \dots, \mathbf{X}_N$  from the density  $f(\cdot; \mathbf{v}_{t-1})$  and compute the sample  $(1 - \varrho)$ -quantile  $\hat{\gamma}_t$  of the performances according to (2.23), provided  $\hat{\gamma}_t$  is less than  $\gamma$ . Otherwise set  $\hat{\gamma}_t = \gamma$ .
3. Use the **same** sample  $\mathbf{X}_1, \dots, \mathbf{X}_N$  to solve the stochastic program (2.25). 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  $\ell$  using the LR estimate

$$\hat{\ell} = \frac{1}{N} \sum_{i=1}^N I_{\{S(\mathbf{X}_i) \geq \gamma\}} W(\mathbf{X}_i; \mathbf{u}, \hat{\mathbf{v}}_T), \quad (2.26)$$

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

*Example 2.2.* We return to the example in Section 2.2.1. In this case, from (2.1) we have

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

so that the  $j$ -th equation of (2.20) becomes

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

therefore

$$v_j = \frac{\sum_{i=1}^N I_{\{S(\mathbf{X}_i) \geq \gamma\}} W(\mathbf{X}_i; \mathbf{u}, \mathbf{w}) X_{ij}}{\sum_{i=1}^N I_{\{S(\mathbf{X}_i) \geq \gamma\}} W(\mathbf{X}_i; \mathbf{u}, \mathbf{w})}, \quad (2.27)$$

which leads to the updating formula (2.8).

Actually, one can show that if the distributions belong to a NEF that is parameterized by the mean, the updating formula always becomes (2.27).

*Remark 2.3 (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 rather refer to [46, 15] and Chapter 3. The main point is that Algorithm 2.3.1 holds true without much alteration.

## 2.4 The CE-Method for Combinatorial Optimization

In this section we discuss the main ideas behind the CE algorithm for combinatorial optimization. When reading this section, the reader is encouraged to refer to the toy example in Section 2.2.2.

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  $\gamma^*$ . Thus,

$$S(\mathbf{x}^*) = \gamma^* = \max_{\mathbf{x} \in \mathcal{X}} S(\mathbf{x}) . \quad (2.28)$$

The starting point in the methodology of the CE method is to associate with the optimization problem (2.28) a meaningful *estimation problem*. To this end we define a collection of indicator functions  $\{I_{\{S(\mathbf{x}) \geq \gamma\}}\}$  on  $\mathcal{X}$  for various levels  $\gamma \in \mathbb{R}$ . Next, let  $\{f(\cdot; \mathbf{v}), \mathbf{v} \in \mathcal{V}\}$  be a family of (discrete) probability densities on  $\mathcal{X}$ , parameterized by a real-valued parameter (vector)  $\mathbf{v}$ . For a certain  $\mathbf{u} \in \mathcal{V}$  we associate with (2.28) the problem of estimating the number

$$\ell(\gamma) = \mathbb{P}_{\mathbf{u}}(S(\mathbf{X}) \geq \gamma) = \sum_{\mathbf{x}} I_{\{S(\mathbf{x}) \geq \gamma\}} f(\mathbf{x}; \mathbf{u}) = \mathbb{E}_{\mathbf{u}} I_{\{S(\mathbf{X}) \geq \gamma\}} , \quad (2.29)$$

where  $\mathbb{P}_{\mathbf{u}}$  is the probability measure under which the random state  $\mathbf{X}$  has pdf  $f(\cdot; \mathbf{u})$ , and  $\mathbb{E}_{\mathbf{u}}$  denotes the corresponding expectation operator. We will call the estimation problem (2.29) the *associated stochastic problem* (ASP). To indicate how (2.29) is associated with (2.28), suppose for example that  $\gamma$  is equal to  $\gamma^*$  and that  $f(\cdot; \mathbf{u})$  is the uniform density on  $\mathcal{X}$ . Note that, typically,  $\ell(\gamma^*) = f(\mathbf{x}^*; \mathbf{u}) = 1/|\mathcal{X}|$  — 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 (2.26) with reference parameter  $\mathbf{v}^*$  given by

$$\mathbf{v}^* = \underset{\mathbf{v}}{\operatorname{argmax}} \mathbb{E}_{\mathbf{u}} I_{\{S(\mathbf{X}) \geq \gamma\}} \ln f(\mathbf{X}; \mathbf{v}) . \quad (2.30)$$

This parameter could be estimated by

$$\widehat{\mathbf{v}^*} = \underset{\mathbf{v}}{\operatorname{argmax}} \frac{1}{N} \sum_{i=1}^N I_{\{S(\mathbf{x}_i) \geq \gamma\}} \ln f(\mathbf{X}_i; \mathbf{v}) , \quad (2.31)$$

where the  $\mathbf{X}_i$  are generated from pdf  $f(\cdot; \mathbf{u})$ . It is plausible that, if  $\gamma$  is close to  $\gamma^*$ , that  $f(\cdot; \mathbf{v}^*)$  assigns most of its probability mass close to  $\mathbf{x}^*$ , and thus can be used to generate an approximate solution to (2.28). However, it is important to note that the estimator (2.31) is only of practical use when  $I_{\{S(\mathbf{x}) \geq \gamma\}} = 1$  for enough samples. This means for example that when  $\gamma$  is close to  $\gamma^*$ ,  $\mathbf{u}$  needs to be such that  $\mathbb{P}_{\mathbf{u}}(S(\mathbf{X}) \geq \gamma)$  is not too small. Thus, the choice of  $\mathbf{u}$  and  $\gamma$  in (2.28) are closely related. On the one hand we would like to choose  $\gamma$  as close as possible to  $\gamma^*$ , 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  $\gamma$  relative large 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 2.3. The idea, based essentially on Algorithm 2.3.1, is to adopt a two-phase multilevel approach in which we simultaneously construct a *sequence* of levels  $\hat{\gamma}_1, \hat{\gamma}_2, \dots, \hat{\gamma}_T$  and parameter (vectors)  $\hat{\mathbf{v}}_0, \hat{\mathbf{v}}_1, \dots, \hat{\mathbf{v}}_T$  such that  $\hat{\gamma}_T$  is close to the optimal  $\gamma^*$  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 for example [145]:

**Algorithm 2.4.1 (Main CE Algorithm for Optimization)**

1. Define  $\hat{\mathbf{v}}_0 = \mathbf{u}$ . Set  $t = 1$  (level counter).
2. Generate a sample  $\mathbf{X}_1, \dots, \mathbf{X}_N$  from the density  $f(\cdot; \mathbf{v}_{t-1})$  and compute the sample  $(1 - \varrho)$ -quantile  $\hat{\gamma}_t$  of the performances according to (2.23).
3. Use the **same** sample  $\mathbf{X}_1, \dots, \mathbf{X}_N$  and solve the stochastic program (2.25) with  $W = 1$ . Denote the solution by  $\hat{\mathbf{v}}_t$ .
4. If for some  $t \geq d$ , say  $d = 5$ ,

$$\hat{\gamma}_t = \hat{\gamma}_{t-1} = \dots = \hat{\gamma}_{t-d}, \quad (2.32)$$

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

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

*Remark 2.4 (Smoothed Updating).* Instead of updating the parameter vector  $\hat{\mathbf{v}}_{t-1}$  to  $\hat{\mathbf{v}}_t$  directly via (2.31) we use a *smoothed* updating procedure in which

$$\hat{\mathbf{v}}_t = \alpha \hat{\mathbf{w}}_t + (1 - \alpha) \hat{\mathbf{v}}_{t-1}, \quad (2.33)$$

where  $\hat{\mathbf{w}}_t$  is the vector derived via (2.25) 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  $\alpha$  between  $0.3 \leq \alpha \leq 0.9$  gives good results. Clearly for  $\alpha = 1$  we have the original updating rule in Algorithm 2.4.1.

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

*Remark 2.5 (Similarities and differences).* Despite the great similarity between Algorithm 2.3.1 and Algorithm 2.4.1, it is important to note that the role of the initial reference parameter  $\mathbf{u}$  is significantly different. In Algorithm 2.3.1  $\mathbf{u}$  is the unique *nominal* parameter for estimating  $\mathbb{P}_{\mathbf{u}}(S(\mathbf{X}) \geq \gamma)$ . However, in Algorithm 2.4.1 the choice for the initial parameter  $\mathbf{u}$  is fairly arbitrary; it is only used to define the ASP. In contrast to Algorithm 2.3.1 the ASP for Algorithm 2.4.1 is redefined after each iteration. In particular, in Steps 2 and 3 of Algorithm 2.4.1 we determine the optimal reference parameter associated with  $\mathbb{P}_{\hat{\mathbf{v}}_{t-1}}(S(\mathbf{X}) \geq \hat{\gamma}_t)$ , instead of  $\mathbb{P}_{\mathbf{u}}(S(\mathbf{X}) \geq \hat{\gamma}_t)$ . Consequently, the likelihood ratio term  $W$  that plays a crucial role in Algorithm 2.3.1 does not appear in Algorithm 2.4.1.

The above procedure can, in principle, be applied to any discrete and continuous optimization problem. 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 densities  $\{f(\cdot; \mathbf{v})\}$ .
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 2.6.* We return to the example from Section 2.2.2. In this case, the random vector  $\mathbf{X} = (X_1, \dots, X_n) \sim \text{Ber}(\mathbf{p})$ , and the parameter vector  $\mathbf{v}$  is  $\mathbf{p}$ . Consequently, the pdf is

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

and since each  $X_j$  can only be 0 or 1,

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

Now we can find the maximum in (2.25) (with  $W = 1$ ) by setting the first derivatives with respect to  $p_j$  equal to zero, for  $j = 1, \dots, n$ :

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

Thus, we get

$$p_j = \frac{\sum_{i=1}^N I_{\{S(\mathbf{x}_i) \geq \gamma\}} X_{ij}}{\sum_{i=1}^N I_{\{S(\mathbf{x}_i) \geq \gamma\}}} , \quad (2.34)$$

which immediately implies (2.10).

A number of remarks are now in order.

*Remark 2.7 (Single-Phase Versus Two-Phase Approach).* Algorithm 2.4.1 is a *two-phase* algorithm. That is, at each iteration  $t$  both the reference parameter  $\mathbf{v}_t$  and the level parameter  $\gamma_t$  are updated. It is not difficult, using the same rationale as before, to formulate a *single-phase* CE algorithm. In particular, consider maximization problem (2.28). Let  $\varphi$  be any increasing “reward”-function of the performances. Let  $\{f(\cdot; \mathbf{v})\}$  be a family of densities on  $\mathcal{X}$  which contains the Dirac measure at  $\mathbf{x}^*$ . Then, solving problem (2.28) is equivalent to solving

$$\max_{\mathbf{v}} \mathbb{E}_{\mathbf{v}} \varphi(S(\mathbf{X})) ,$$

or solving

$$\max_{\mathbf{v}} \mathbb{E}_{\mathbf{u}} \varphi(S(\mathbf{X})) \ln f(\mathbf{X}; \mathbf{v}) ,$$

for any  $\mathbf{u}$ . As before we construct a sequence of parameter vectors  $\hat{\mathbf{v}}_0 = \mathbf{u}, \hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \dots$ , such that

$$\hat{\mathbf{v}}_t = \operatorname{argmax}_{\mathbf{v}} \sum_{i=1}^N \varphi(S(\mathbf{X}_i)) \ln f(\mathbf{X}_i; \mathbf{v}) ,$$

where  $\mathbf{X}_1, \dots, \mathbf{X}_N$  is a random sample from  $f(\cdot; \hat{\mathbf{v}}_{t-1})$ . A reward function without a level parameter  $\gamma$  would simplify Algorithm 2.4.1 substantially. The disadvantage of using this approach is that, typically, it takes too long for Algorithm 2.4.1 to converge, since the large number of “not important” vectors slow down dramatically the convergence of  $\{\hat{\mathbf{v}}_t\}$  to  $\mathbf{v}^*$  corresponding to the Dirac measure at  $\mathbf{x}^*$ . We found numerically that the single-phase CE algorithm is much worse than its two-phase counterpart in Algorithm 2.4.1, in the sense that it is less accurate and more time consuming. Hence it is *crucial* for the CE method to use both phases, that is, follow Algorithm 2.4.1. This is also one of the major differences between CE and ant-based methods, where a single-phase procedure (updating of  $\hat{\mathbf{v}}_t$  alone, no updating of  $\hat{\gamma}_t$ ) is used [49].

*Remark 2.8 (Maximum Likelihood Estimation).* It is interesting to note the connection between (2.25) and *maximum likelihood estimation* (MLE). In the MLE problem we are given data  $\mathbf{x}_1, \dots, \mathbf{x}_N$  which are thought to be the outcomes of i.i.d. random variables  $\mathbf{X}_1, \dots, \mathbf{X}_N$  (random sample) each having a distribution  $f(\cdot; \mathbf{v})$ , where the parameter (vector)  $\mathbf{v}$  is an element of some set  $\mathcal{V}$ . We wish to estimate  $\mathbf{v}$  on the basis of the data  $\mathbf{x}_1, \dots, \mathbf{x}_N$ . The *maximum likelihood estimate* (MLE) is that parameter  $\hat{\mathbf{v}}$  which maximizes the joint density of  $\mathbf{X}_1, \dots, \mathbf{X}_N$  for the given data  $\mathbf{x}_1, \dots, \mathbf{x}_N$ . In other words,

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

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



$$\hat{\mathbf{v}} = \operatorname{argmax}_{\mathbf{v}} \sum_{i=1}^N \ln f(\mathbf{X}_i; \mathbf{v}) . \quad (2.35)$$

Solving (2.35) is similar to solving (2.25). The only differences are the indicator function  $I_{\{S(\mathbf{X}_i) \geq \gamma\}}$  and the likelihood ratio  $W$ . For  $W = 1$  we can write Step 3 in Algorithm 2.4.1 as

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

In other words,  $\hat{\mathbf{v}}_t$  is equal to the MLE of  $\hat{\mathbf{v}}_{t-1}$  based only on the vectors  $\mathbf{X}_i$  in the random sample for which the performance is greater than or equal to  $\hat{\gamma}_t$ . For example, in Example 2.6 the MLE of  $p_j$  based on a random sample  $\mathbf{X}_1, \dots, \mathbf{X}_N$  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  $\gamma$ , we obtain (2.34) immediately.

*Remark 2.9 (Parameters).* The choice for the sample size  $N$  and the parameter  $\varrho$  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  $N = cn$ , 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  $N = cn^2$ , where  $n^2$  is the number of *edges* in the network. It is crucial to realize that the sample sizes  $N = cn$  and  $N = cn^2$  (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 estimate  $k$  parameters reliably, one needs to take *at least* a sample  $N = ck$  for some constant  $c > 1$ . Regarding  $\varrho$ , it is suggested to take  $\varrho$  around 0.01, provided  $n$  is reasonably large, say  $n \geq 100$ ; and it is suggested to take a larger  $\varrho$ , say  $\varrho \approx \ln(n)/n$ , for  $n < 100$ .

Alternatively, the choice of  $N$  and  $\varrho$  can be determined *adaptively*. For example, in [85] an adaptive algorithm is proposed that adjusts  $N$  automatically. The FACE algorithm discussed in Chapter 5 is another option.

## 2.5 Two Applications

In this section we discuss the application of the CE method to two combinatorial optimization problems, namely the max-cut problem, which is a typical SNN-type problem; and the travelling salesman problem, which is a typical SEN-type 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. For a more in-depth study of the max-cut problem and the related *partition problem* we refer to Sections 4.5 and 4.6. Similarly, the TSP is discussed in more detail in Section 4.7.

### 2.5.1 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  $V = \{1, \dots, n\}$  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 nonnegative. 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 nonnegative, 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  $V = \{1, \dots, 6\}$ , then  $\{\{1, 3, 4\}, \{2, 5, 6\}\}$  is a possible cut. The *cost* of a cut is sum of the weights across the cut. As an example, consider the following  $6 \times 6$  cost matrix

$$C = \begin{pmatrix} 0 & c_{12} & c_{13} & 0 & 0 & 0 \\ c_{21} & 0 & c_{23} & c_{24} & 0 & 0 \\ c_{31} & c_{32} & 0 & c_{34} & c_{35} & 0 \\ 0 & c_{42} & c_{43} & 0 & c_{45} & c_{46} \\ 0 & 0 & c_{53} & c_{54} & 0 & c_{56} \\ 0 & 0 & 0 & c_{64} & c_{65} & 0 \end{pmatrix}. \quad (2.36)$$

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

$$c_{12} + c_{32} + c_{42} + c_{45} + c_{53} + c_{46}.$$

It will be convenient to represent a cut via a *cut vector*  $\mathbf{x} = (x_1, \dots, x_n)$ , where  $x_i = 1$  if node  $i$  belongs to same partition as 1, and 0 otherwise. By definition  $x_1 = 1$ . For example, the cut  $\{\{1, 3, 4\}, \{2, 5, 6\}\}$  can be represented via the cut vector  $(1, 0, 1, 1, 0, 0)$ .

Let  $\mathcal{X}$  be the set of all cut vectors  $\mathbf{x} = (1, x_2, \dots, x_n)$  and let  $S(\mathbf{x})$  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  $X_2, \dots, X_n$  be independent Bernoulli random variables with success probabilities  $p_2, \dots, p_n$ , exactly as in the second toy example; see Section 2.2.2.

It immediately follows (see Example 2.6) that the updating formula in Algorithm 2.4.1 at the  $t$ -th iteration is given by

$$\hat{p}_{t,i} = \frac{\sum_{k=1}^N I_{\{S(\mathbf{x}_k) \geq \gamma_t\}} I_{\{X_{ki}=1\}}}{\sum_{k=1}^N I_{\{S(\mathbf{x}_k) \geq \gamma_t\}}}, \quad (2.37)$$

$i = 2, \dots, n$ .

### A Synthetic Max-Cut Problem

Since the max-cut problem is NP hard [57, 125], 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  $m \in \{1, \dots, n\}$  consider the following symmetric cost matrix:

$$C = \begin{pmatrix} Z_{11} & B_{12} \\ B_{21} & Z_{22} \end{pmatrix}, \quad (2.38)$$

where  $Z_{11}$  is an  $m \times m$  symmetric matrix in which all the upper-diagonal elements are generated from a  $U(a, b)$  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  $c > b(n - m)/m$  the optimal cut is given, by construction, by  $V^* = \{V_1^*, V_2^*\}$ , with

$$V_1^* = \{1, \dots, m\} \quad \text{and} \quad V_2^* = \{m + 1, \dots, n\}, \quad (2.39)$$

and the optimal value of the cut is

$$\gamma^* = cm(n - m). \quad (2.40)$$

Of course a similar 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  $c$ .

Table 2.3 lists a typical output of Algorithm 2.4.1 applied to the synthetic max-cut problem, for a network with  $n = 400$  nodes. The convergence of the reference vectors  $\{\mathbf{p}_t\}$  to the optimal  $\mathbf{p}^*$  is further illustrated in Figures 2.3 and 2.4.

In this table besides the  $(1 - \varrho)$ -quantile of the performances  $\widehat{\gamma}_t$ , we also list the *best* of the performances in each iteration, denoted by  $S_{t,(N)}$ , and the Euclidean distance

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

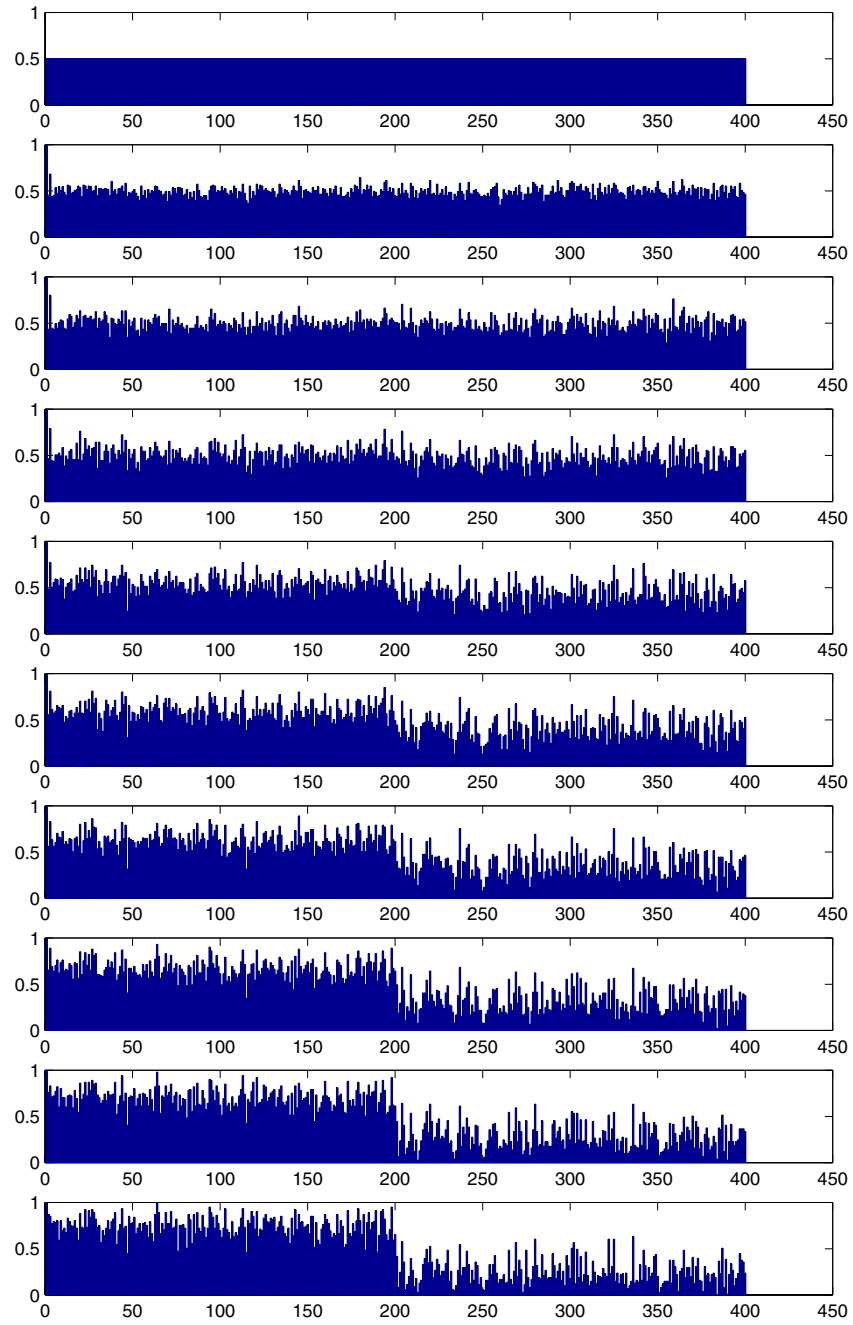
as a measure of how close the reference vector is to the optimal reference vector  $\mathbf{p}^* = (1, 1, \dots, 1, 0, 0, \dots, 0)$ .

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

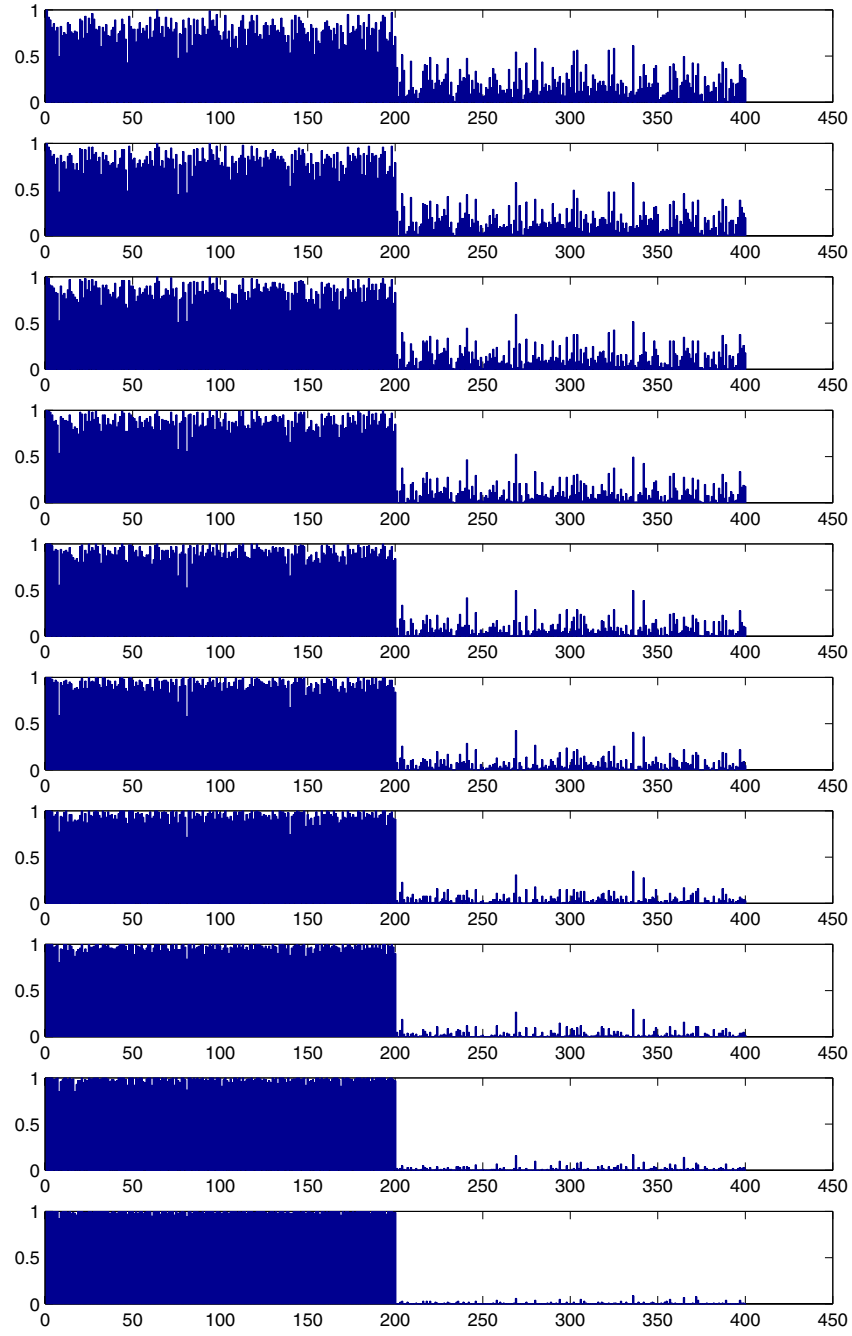
The CPU time was only 100 seconds, using a Matlab implementation on a Pentium III, 500 Mhz processor. We see that the CE algorithm converges quickly, yielding the exact optimal solution 40000 in less than 23 iterations.

**Table 2.3.** A typical evolution of Algorithm 2.4.1 for the synthetic max-cut problem with  $n = 400$ ,  $d = 3$ ,  $\varrho = 0.1$ ,  $\alpha = 1.0$ ,  $N = 1000$ .

$t$	$\widehat{\gamma}_t$	$S_{t,(N)}$	$\ \widehat{\mathbf{p}}_t - \mathbf{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



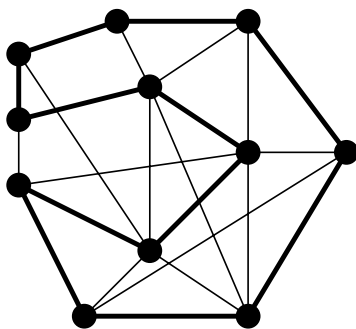
**Fig. 2.3.** Sequence of reference vectors for the synthetic max-cut problem with 400 nodes. Iterations  $0, 1, \dots, 9$ .



**Fig. 2.4.** Sequence of reference vectors for the synthetic max-cut problem with 400 nodes. Iterations 10, ..., 19.

### 2.5.2 The Travelling Salesman Problem

The travelling salesman problem (TSP) can be formulated as follows: Consider a weighted graph  $G$  with  $n$  nodes, labeled  $1, 2, \dots, n$ . 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<sup>†</sup> (except the starting city, which is also the terminating city); see Figure 2.5.



**Fig. 2.5.** Find the shortest tour  $\mathbf{x}$  visiting all nodes.

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  $S(\mathbf{x})$  the total length of tour  $\mathbf{x} \in \mathcal{X}$ . We can represent each tour via a *permutation* of  $(1, \dots, n)$ . For example for  $n = 4$ , the permutation  $(1, 3, 2, 4)$  represents the tour  $1 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 1$ . In fact, we may as well represent a tour via a permutation  $\mathbf{x} = (x_1, \dots, x_n)$  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.

$$\min_{\mathbf{x} \in \mathcal{X}} S(\mathbf{x}) = \min_{\mathbf{x} \in \mathcal{X}} \left\{ \sum_{i=1}^{n-1} c_{x_i, x_{i+1}} + c_{x_n, 1} \right\}. \quad (2.41)$$

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

$$|\mathcal{X}| = (n-1)! \quad (2.42)$$

This is exactly the setting of Section 2.4, so we can use the CE method to solve (2.41). Note however that we need to modify Algorithm 2.4.1 since we have here a *minimization* problem.

<sup>†</sup> In some versions of the TSP, cities can be visited more than once.

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 (2.41) to an *equivalent* minimization problem. Let

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

be the set of vectors that correspond to paths of length  $n$  that start 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  $\mathbf{x} = (1, 3, 1, 3) \in \tilde{\mathcal{X}}$ , corresponding to the path  $1 \rightarrow 3 \rightarrow 1 \rightarrow 3 \rightarrow 1$ . Define the function  $\tilde{S}$  on  $\tilde{\mathcal{X}}$  by  $\tilde{S}(\mathbf{x}) = S(\mathbf{x})$ , if  $\mathbf{x} \in \mathcal{X}$  and  $\tilde{S}(\mathbf{x}) = \infty$ , otherwise. Then, obviously (2.41) is equivalent to the minimization problem

$$\text{minimize } \tilde{S}(\mathbf{x}) \text{ over } \mathbf{x} \in \tilde{\mathcal{X}}. \quad (2.43)$$

A simple method to generate a random path  $\mathbf{X} = (X_1, \dots, X_n)$  in  $\tilde{\mathcal{X}}$  is to use a Markov chain on 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  $\mathbf{X}$  is thus parameterized by the matrix  $P$  and its logarithm is given by

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

where  $\tilde{\mathcal{X}}_{ij}(r)$  is the set of all paths in  $\tilde{\mathcal{X}}$  for which the  $r$ -th transition is from node  $i$  to  $j$ . The updating rules for this modified optimization problem follow from (2.25) ( $W = 1$ ), with  $\{S(\mathbf{X}_i) \geq \gamma\}$  replaced with  $\{\tilde{S}(\mathbf{X}_i) \leq \gamma\}$ , under the condition that the rows of  $P$  sum up to 1. Using Lagrange multipliers  $u_1, \dots, u_n$  we obtain the maximization problem

$$\max_P \min_{u_1, \dots, u_n} \left[ \mathbb{E}_P I_{\{\tilde{S}(\mathbf{X}) \leq \gamma\}} \ln f(\mathbf{X}; P) + \sum_{i=1}^n u_i \left( \sum_{j=1}^n p_{ij} - 1 \right) \right]. \quad (2.44)$$

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

$$\frac{\mathbb{E}_P I_{\{\tilde{S}(\mathbf{X}) \leq \gamma\}} \sum_{r=1}^n I_{\{\mathbf{x} \in \tilde{\mathcal{X}}_{ij}(r)\}}}{p_{ij}} + u_i = 0. \quad (2.45)$$

Summing over  $j = 1, \dots, n$  gives  $\mathbb{E}_P I_{\{\tilde{S}(\mathbf{X}) \leq \gamma\}} \sum_{r=1}^n I_{\{\mathbf{x} \in \tilde{\mathcal{X}}_i(r)\}} = -u_i$ , 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



$$p_{ij} = \frac{\mathbb{E}_P I_{\{\tilde{S}(\mathbf{X}) \leq \gamma\}} \sum_{r=1}^n I_{\{\mathbf{X} \in \tilde{\mathcal{X}}_{ij}(r)\}}}{\mathbb{E}_P I_{\{\tilde{S}(\mathbf{X}) \leq \gamma\}} \sum_{r=1}^n I_{\{\mathbf{X} \in \tilde{\mathcal{X}}_i(r)\}}} . \quad (2.46)$$

The corresponding estimator is

$$\hat{p}_{ij} = \frac{\sum_{k=1}^N I_{\{\tilde{S}(\mathbf{X}_k) \leq \gamma\}} \sum_{r=1}^n I_{\{\mathbf{X}_k \in \tilde{\mathcal{X}}_{ij}(r)\}}}{\sum_{k=1}^N I_{\{\tilde{S}(\mathbf{X}_k) \leq \gamma\}} \sum_{r=1}^n I_{\{\mathbf{X}_k \in \tilde{\mathcal{X}}_i(r)\}}} . \quad (2.47)$$

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 into account only those paths that have a total length less than or equal to  $\gamma$ .

This is how we could, *in principle*, carry out the sample generation and parameter updating for problem (2.43). We generate the path via a Markov process with transition matrix  $P$ , and use updating formula (2.47). However, *in practice*, we would never generate the paths in this way, since the majority of these tours would be irrelevant since they would not constitute a tour, and therefore their  $\tilde{S}$  values would be  $\infty$ . In order to avoid the generation of irrelevant tours, we proceed as follows.

**Algorithm 2.5.1 ( 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  $P^{(k+1)}$ .
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 2.5.1 we are merely *speeding up* our naive way of generating the paths. Moreover, since we now only generate *tours*, the updated value for  $p_{ij}$  can be estimated as

$$\hat{p}_{ij} = \frac{\sum_{k=1}^N I_{\{S(\mathbf{X}_k) \leq \gamma\}} I_{\{\mathbf{X}_k \in \mathcal{X}_{ij}\}}}{\sum_{k=1}^N I_{\{S(\mathbf{X}_k) \leq \gamma\}}} , \quad (2.48)$$

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 (2.47).

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 (2.32).

### 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 **ft53** taken from the URL

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

Table 2.4 presents a typical evolution of the CE Algorithm for the problem **ft53**, which defines an asymmetric fully connected graph of size 53, where the cost of each edge  $c_{ij}$  is given. The CE parameters were: stopping parameter  $d = 5$ , rarity parameter  $\varrho = 0.01$ , sample size  $N = 10n^2 = 28090$ , and smoothing parameter  $\alpha = 0.7$ . The *relative experimental error* of the solution is

$$\varepsilon = \frac{\hat{\gamma}_T - \hat{\gamma}^*}{\hat{\gamma}^*} = 0.015, \quad (2.49)$$

where  $\hat{\gamma}^* = 6905$  is the best known solution. The CPU time was approximately 6 minutes. In Table 2.4,  $S_{t,(1)}$  denotes the length of smallest tour in iteration  $t$ . We also included the quantity  $P_t^{mm}$ , which is the minimum of the maximum elements in each row of matrix  $\hat{P}_t$ .

**Table 2.4.** A typical evolution of Algorithm 2.4.1 for the TSP **ft53** with  $n = 53$  nodes,  $d = 5$ ,  $\varrho = 0.01$ ,  $\alpha = 0.7$ ,  $N = 10n^2 = 28090$ .

$t$	$\hat{\gamma}_t$	$S_{t,(1)}$	$P_t^{mm}$
1	23234.00	21111.00	0.0354
2	20611.00	18586.00	0.0409
3	18686.00	16819.00	0.0514
4	17101.00	14890.00	0.0465
5	15509.00	13459.00	0.0698
6	14449.00	12756.00	0.0901
7	13491.00	11963.00	0.0895
8	12773.00	11326.00	0.1065
9	12120.00	10357.00	0.0965
10	11480.00	10216.00	0.1034
11	11347.00	9952.00	0.1310
12	10791.00	9525.00	0.1319
13	10293.00	9246.00	0.1623
14	10688.00	9176.00	0.1507
15	9727.00	8457.00	0.1346
16	9263.00	8424.00	0.1436

$t$	$\hat{\gamma}_t$	$S_{t,(1)}$	$P_t^{mm}$
17	9422.00	8614.00	0.1582
18	9155.00	8528.00	0.1666
19	8661.00	7970.00	0.1352
20	8273.00	7619.00	0.1597
21	8096.00	7485.00	0.1573
22	7868.00	7216.00	0.1859
23	7677.00	7184.00	0.2301
24	7519.00	7108.00	0.2421
25	7420.00	7163.00	0.2861
26	7535.00	7064.00	0.3341
27	7506.00	7072.00	0.3286
28	7199.00	7008.00	0.3667
29	7189.00	7024.00	0.3487
30	7077.00	7008.00	0.4101
31	7068.00	7008.00	0.4680

Similar performances were found for other TSPs in the benchmark library above. Table 2.5 presents the performance of Algorithm 2.4.1 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  $\varrho = 10^{-2}$ ,  $N = 10n^2$ ,  $\alpha = 0.7$  (smoothing parameter in (2.33)) and  $d = 5$  (in (2.32)). To study the variability in the solutions, each problem was repeated 10 times. In Table 2.5,  $n$  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 final estimates of the optimal solution,  $\gamma^*$  denotes the best known solution,  $\bar{\varepsilon}$  denotes the average relative experimental error based on 10 replications,  $\varepsilon_*$  and  $\varepsilon^*$  denote the smallest and the largest relative error among the 10 generated shortest paths, 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  $\varepsilon$  in Table 2.5 increase at most by a factor of 1.5.

**Table 2.5.** Case studies for the TSP.

file	$n$	$\bar{T}$	$\bar{\gamma}_1$	$\bar{\gamma}_T$	$\gamma^*$	$\bar{\varepsilon}$	$\varepsilon_*$	$\varepsilon^*$	CPU
<b>br17</b>	17	23.8	68.2	39.0	39	0.000	0.000	0.000	9
<b>ftv33</b>	34	31.2	3294.0	1312.2	1286	0.020	0.000	0.062	73
<b>ftv35</b>	36	31.5	3714.0	1490.0	1473	0.012	0.004	0.018	77
<b>ftv38</b>	39	33.8	4010.8	1549.8	1530	0.013	0.004	0.032	132
<b>p43</b>	43	44.5	9235.5	5624.5	5620	0.010	0.000	0.001	378
<b>ftv44</b>	45	35.5	4808.2	1655.8	1613	0.027	0.013	0.033	219
<b>ftv47</b>	48	40.2	5317.8	1814.0	1776	0.021	0.006	0.041	317
<b>ry48p</b>	48	40.8	40192.0	14845.5	14422	0.029	0.019	0.050	345
<b>ft53</b>	53	39.5	20889.5	7103.2	6905	0.029	0.025	0.035	373
<b>ftv55</b>	56	40.0	5835.8	1640.0	1608	0.020	0.002	0.043	408
<b>ftv64</b>	65	43.2	6974.2	1850.0	1839	0.006	0.000	0.014	854
<b>ftv70</b>	71	47.0	7856.8	1974.8	1950	0.013	0.004	0.037	1068
<b>ft70</b>	70	42.8	64199.5	39114.8	38673	0.011	0.003	0.019	948

## Dynamics

Finally, as an illustration of the dynamics of the CE algorithm, we display below the sequence of matrices  $\hat{P}_0, \hat{P}_1, \dots$  for a TSP with  $n=10$  cities, where the optimal tour is  $(1, 2, 3, \dots, 10, 1)$ . A graphical illustration of the convergence is given in Figure 2.6, where we omitted  $\hat{P}_0$  whose off-diagonal elements are all equal to  $1/9$  and diagonal elements equal to 0.

$$\begin{aligned}
\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} .
\end{aligned}$$

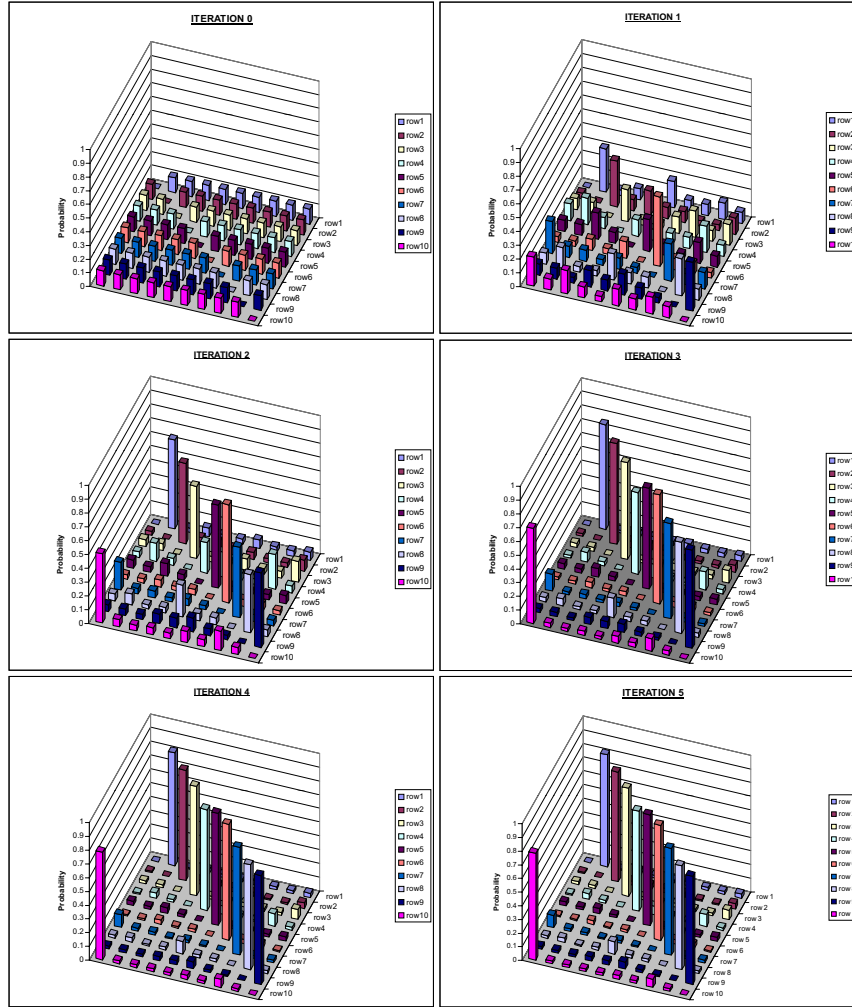


Fig. 2.6. Convergence of the reference parameter (matrix) for a 10 node TSP.

## 2.6 Exercises

1. Implement and repeat the rare-event simulation toy example corresponding to Table 2.1.
2. Implement and repeat the combinatorial optimization toy example corresponding to Table 2.2.
3. Extend the program used in Exercise 2 to include smoothed updating, and apply the program to a larger example, say  $n = 50$ . Observe if and how the choice of parameters affects the accuracy and speed of the algorithm.

4. Consider the one-phased analog of the two-phased CE program in Exercise 3; see Remark 2.7. Use the reward function  $\varphi(s) = s$ .

a) Show that the updating formulas for the  $p_j$ 's become:

$$\hat{p}_{t,j} = \frac{\sum_{i=1}^N S(\mathbf{X}_i) X_{ij}}{\sum_{i=1}^N S(\mathbf{X}_i)}.$$

b) Compare numerically the performances of the one- and two-phased algorithms.

5. Verify (2.40) and show that  $c > b(n-m)m$  is a sufficient condition for  $V^*$  in (2.39) to be the optimal cut.
6. In the famous *n-queens problem*, introduced by Gauss in 1850, the objective is to position  $n$  queens on an  $n \times n$  chess board such that no one queen can be taken by any other. Write a computer program that solves the 8-queens problem using the CE method. We may assume that the queens occupy different rows. The positions of the queens may thus be represented as a vector  $\mathbf{x} \in \{1, \dots, 8\}^8$ , where  $x_i$  indicates the column that the queen of row  $i$  occupies. For example, the configuration of Figure 2.7 corresponds to  $\mathbf{x} = (2, 3, 7, 4, 8, 5, 1, 6)$ . A straightforward way to generate random vectors  $\mathbf{X}$  is to draw each  $X_i$  independently from a probability vector  $(p_{i1}, \dots, p_{i8})$ ,  $i = 1, \dots, 8$ . The performance function  $S$  could be chosen such that it represent that number of times the queens can attack each other. That is, the sum of the number of queens minus one, in each row, column and diagonal. In Figure 2.7,  $S(\mathbf{x}) = 1$ . The updating formulas for the  $p_{ij}$  are easy. Excluding symmetry, there are 12 different solutions to the problem. Find them all, by running your algorithm several times. Take  $N = 500$ ,  $\alpha = 0.7$  and  $\varrho = 0.1$ .

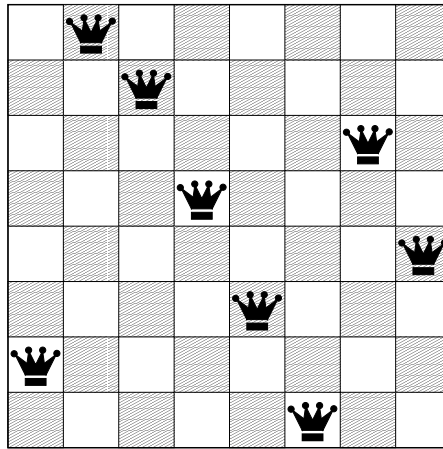


Fig. 2.7. Position the 8 queens such that no queen can attack another.

The Cross-Entropy Method

A Unified Approach to Combinatorial Optimization,  
Monte-Carlo Simulation and Machine Learning

Rubinstein, R.Y.; Kroese, D.P.

2004, XX, 301 p. 60 illus. With online files/update.,

Hardcover

ISBN: 978-0-387-21240-1