

A Tutorial on the Cross-Entropy Method

Gabriele Martinelli

March 10, 2009

From a paper by
De Boer, Kroese, Mannor and Rubinstein

Index

- 1 Introduction
- 2 Two examples
 - Rare event simulation
 - Combinatorial Optimization
- 3 The main algorithm
 - The CE method for rare events simulation
 - The CE method for Combinatorial Optimization
- 4 Applications
 - The max-cut problem
 - The traveling salesman problem
 - The Markovian Decision Process
 - Conclusions

1 Introduction

2 Two examples

- Rare event simulation
- Combinatorial Optimization

3 The main algorithm

- The CE method for rare events simulation
- The CE method for Combinatorial Optimization

4 Applications

- The max-cut problem
- The traveling salesman problem
- The Markovian Decision Process
- Conclusions

Definition

Cross-entropy is a *simple, efficient* and *general* method for solving optimization problems:

- Combinatorial optimization problems (COP), where the problem is known and static (f.i. The travelling salesman problem (TSP) and the max-cut problem)
- Buffer allocation problem (BAP), where the objective function needs to be estimated since it is unknown.
- Rare event simulation (f.i. in reliability or telecommunication systems)

Key-point

Deterministic optimization problem \Rightarrow Stochastic optimization problem

How does it work

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

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

How does it work

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

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

Other similar algorithms

The same idea is used in other well-known randomized methods for combinatorial optimization problems, such these:

- Simulated annealing
- Genetic algorithms
- Nested partitioning method
- Ant colony optimization

The name

The method derives its name from the cross-entropy (or Kullback-Leibler) distance, a well known measure of "information"; this distance is used in one fundamental step of the algorithm.

The name

The method derives its name from the cross-entropy (or Kullback-Leibler) distance, a well known measure of "information"; this distance is used in one fundamental step of the algorithm.

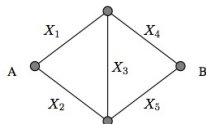
Kinds of CE methods

The problem is formulated as optimization problems concerning a weighted graph, with two possible sources of randomness:

- in the nodes \Rightarrow Stochastic node networks (SNN) (max-cut problem)
- in the edges \Rightarrow Stochastic edge networks (SEN) (travelling salesman problem)

- 1 Introduction
- 2 Two examples**
 - Rare event simulation
 - **Combinatorial Optimization**
- 3 The main algorithm
 - The CE method for rare events simulation
 - The CE method for Combinatorial Optimization
- 4 Applications
 - The max-cut problem
 - The traveling salesman problem
 - The Markovian Decision Process
 - Conclusions

Rare event simulation example



- Weighted graph.
- Random weights X_1, \dots, X_5 . $X_i \sim \mathcal{E}(\frac{1}{u_i})$. So:

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

- $S(\mathbf{X}) =$ length of the shortest path from A to B
- We wish to estimate from simulation

$$l = P(S(\mathbf{X}) > \gamma) = \mathbb{E}(\mathbb{I}_{\{S(\mathbf{x}) > \gamma\}}) = \int (\mathbb{I}_{\{S(\mathbf{x}) > \gamma\}} f(\mathbf{x}; \mathbf{u}) d\mathbf{x}.$$

Methods proposed

- 1 **MC simulation:** draw a random sample and compute

$$\hat{l} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}_{\{S(\mathbf{x}_i) > \gamma\}}.$$

Problem: N has to be very large in order to obtain an accurate estimation of l .

Methods proposed

- 1 **MC simulation:** draw a random sample and compute

$$\hat{l} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}_{\{S(\mathbf{x}_i) > \gamma\}}.$$

Problem: N has to be very large in order to obtain an accurate estimation of l .

- 2 **Importance sampling:** We choose a function $g(\mathbf{x})$ with the same support as $f(\mathbf{x})$, that reduces the requested simulation effort. In this way:

$$l = \int \mathbb{I}_{\{S(\mathbf{x}) > \gamma\}} \frac{f(\mathbf{x})}{g(\mathbf{x})} g(\mathbf{x}) d\mathbf{x}$$

So: $\hat{l} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}_{\{S(\mathbf{x}_i) > \gamma\}} W(\mathbf{X}_i)$, where $W(\mathbf{x}) = \frac{f(\mathbf{x})}{g(\mathbf{x})}$

Problem: How to choose in the best way the distribution $g(\mathbf{x})$?

Methods proposed

- 1 **MC simulation:** draw a random sample and compute

$$\hat{l} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}_{\{S(\mathbf{x}_i) > \gamma\}}.$$

Problem: N has to be very large in order to obtain an accurate estimation of l .

- 2 **Importance sampling:** We choose a function $g(\mathbf{x})$ with the same support as $f(\mathbf{x})$, that reduces the requested simulation effort. In this way:

$$l = \int \mathbb{I}_{\{S(\mathbf{x}) > \gamma\}} \frac{f(\mathbf{x})}{g(\mathbf{x})} g(\mathbf{x}) d\mathbf{x}$$

So: $\hat{l} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}_{\{S(\mathbf{x}_i) > \gamma\}} W(\mathbf{X}_i)$, where $W(\mathbf{x}) = \frac{f(\mathbf{x})}{g(\mathbf{x})}$

Problem: How to choose in the best way the distribution $g(\mathbf{x})$?

If we consider only the simple case when g is also an exponential distribution ($g(x) = \frac{1}{v} e^{-x/v}$), the problem becomes:

Problem: How to choose in the best way the parameters $\mathbf{v} = \{v_1, \dots, v_5\}$?

The CE method is able to give a fast way in order to answer this question.

The CE algorithm

- 1 Define $\mathbf{v}_0 = \mathbf{u}$. Set $t = 1$
- 2 While $\hat{\gamma}_t \neq \gamma$
 - 1 Generate a random sample from $f(\cdot; \mathbf{v}_{t-1})$
 - 2 Calculate the performances $S(\mathbf{X}_i)$ and order them from smallest to biggest.
 - 3 Let $\hat{\gamma}_t := S_{(\lceil(1-\rho)N\rceil)}$, i.e. the $(1 - \rho)$ th quartile, provided that is less than γ . Otherwise, $\hat{\gamma}_t = \gamma$.
 - 4 Calculate for $j = 1, \dots, 5$

$$\hat{\mathbf{v}}_{t,j} = \frac{\sum_{i=1}^N \mathbb{I}_{\{S(\mathbf{x}_i) > \hat{\gamma}_t\}} W(\mathbf{X}_i; \mathbf{u}; \hat{\mathbf{v}}_{t-1}) X_{ij}}{\sum_{i=1}^N \mathbb{I}_{\{S(\mathbf{x}_i) > \hat{\gamma}_t\}} W(\mathbf{X}_i; \mathbf{u}; \hat{\mathbf{v}}_{t-1})}$$

- 3 We estimate l via importance sampling with these new parameters:

$$\hat{l} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}_{\{S(\mathbf{x}_i) > \hat{\gamma}\}} W(\mathbf{X}_i; \mathbf{u}; \hat{\mathbf{v}}_T)$$

Comparison between MC and CE

Example:

- $\mathbf{u} = (0.25, 0.4, 0.1, 0.3, 0.2)$
- We wish to estimate the probability that $S(\mathbf{X}) > 2$. \Rightarrow Rare event!

Crude MonteCarlo:

- 10^7 samples:
 - $\hat{l} = 1.65 \cdot 10^{-5}$
 - Relative error of 0.165
 - 630 sec.
- 10^8 samples:
 - $\hat{l} = 1.30 \cdot 10^{-5}$
 - Relative error of 0.03
 - 6350 sec.

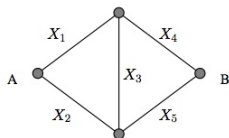
Cross-Entropy method:

- 1000 samples:
 - $\hat{l} = 1.34 \cdot 10^{-5}$
 - Relative error of 0.03
 - 3 sec! + 0.5 sec (5 iterations) in order to get the best parameter vector $\hat{\mathbf{v}}$

Simulation

I tried to simulate the previous example.

- Matrix of the paths:



$$\begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

- MC simulation

```

for i=1:N
    l unh(i)=min(exprnd(u)*tmat);
end
l=length(find(l unh>gamma))/N;
  
```

Results:

- $N = 10^6 : \hat{l}_1 = 0.6 \cdot 10^{-5}, \hat{l}_2 = 1.5 \cdot 10^{-5}, 70 \text{ seconds.}$
- $N = 10^7 : \hat{l}_1 = 1.26 \cdot 10^{-5}, \hat{l}_2 = 1.57 \cdot 10^{-5}, 702 \text{ seconds.}$

Simulation

● CE simulation

```
while gammat<gamma
    for i=1:N
        sampl(i,:)=exprnd(vt);
        lugh(i)=min(sampl(i,:)*mat');
        W(i)=exp(-sampl(i,:)*(1./u-1./vt)')*prod(vt./u);
        %W(x,u,v)=exp(-x*(1./u-1./v)')*prod(v./u);
    end
    gammat=quantile(lugh,(1-rho));
    if gammat>2
        gammat=2;
    end
    vt=(W.*(lugh>=gammat)*sampl)./(sum(W.*(lugh>=gammat)));
    gammat
end

for i=1:N1
    sampl(i,:)=exprnd(vt);
    lugh(i)=min(sampl(i,:)*mat');
    W(i)=exp(-sampl(i,:)*(1./u-1./vt)')*prod(vt./u);
end
l=sum(W.*(lugh>=gammat))/N1;
```

I got the same result as in the paper ($\hat{l} = 1.3377 \cdot 10^{-5}$)

Combinatorial Optimization Example

- Binary vector \mathbf{y} of length n , whose elements are hidden.
- For each vector \mathbf{x} that we can simulate, we get:

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

i.e. the number of matches between \mathbf{x} and \mathbf{y} .

- Our goal is to reconstruct \mathbf{y} , searching for a vector \mathbf{x} that maximizes the function $S(\mathbf{x})$

Combinatorial Optimization Example

- Binary vector \mathbf{y} of length n , whose elements are hidden.
- For each vector \mathbf{x} that we can simulate, we get:

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

i.e. the number of matches between \mathbf{x} and \mathbf{y} .

- Our goal is to reconstruct \mathbf{y} , searching for a vector \mathbf{x} that maximizes the function $S(\mathbf{x})$

Note

The problem actually is trivial: we just have to test all sequences like $(0, 0, \dots, 0)$, $(1, 0, \dots, 0)$, \dots , $(0, 0, \dots, 0, 1)$; we use the problem as a prototype of combinatorial maximization problem

Methods proposed

First approach

Generate repeatedly binary vectors $\mathbf{X} = (X_1, \dots, X_n)$ where X_1, \dots, X_n are independent Bernoulli distribution with parameters p_1, \dots, p_n .

Note: If $\mathbf{p} = \mathbf{y} \Rightarrow P(S(\mathbf{X}) = n) = 1 \Rightarrow$ the optimal sequence is generated with probability 1.

The CE method

Create a sequence $\hat{\mathbf{p}}_0, \hat{\mathbf{p}}_1, \dots$ that converges to the optimal parameter vector $\mathbf{p}^* = \mathbf{y}$.

The convergence is driven by the computation of a performance index $\hat{\gamma}_1, \hat{\gamma}_2, \dots$, that should converge to the optimal performance index $\gamma^* = n$.

The CE algorithm

- 1 Define \mathbf{p}_0 random (or let $\mathbf{p}_0 = (1/2, \dots, 1/2)$). Set $t = 1$
- 2 While stopping criterion
 - 1 Generate a random sample from $f(\cdot; \mathbf{p}_{t-1})$
 - 2 Calculate the performances $S(\mathbf{X}_i)$ and order them from smallest to biggest.
 - 3 Let $\hat{\gamma}_t := S_{(\lceil(1-\rho)N\rceil)}$, i.e. the $(1 - \rho)$ th quartile.
 - 4 Calculate for $j = 1, \dots, n$

$$\hat{p}_{t,j} = \frac{\sum_{i=1}^N \mathbb{I}_{\{S(\mathbf{x}_i) > \hat{\gamma}_t\}} \mathbb{I}_{\{X_{ij}=1\}}}{\sum_{i=1}^N \mathbb{I}_{\{S(\mathbf{x}_i) > \hat{\gamma}_t\}}}$$

A possible stopping criterion is the stationarity of \mathbf{p} for many iterations.

Interpretation of the updating rule: *to update the j -th success probability we count how many vectors of the last sample X_1, \dots, 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 (normalize) this by the number of vectors that have a performance greater than or equal to $\hat{\gamma}_t$.*

- 1 Introduction
- 2 Two examples
 - Rare event simulation
 - Combinatorial Optimization
- 3 The main algorithm**
 - The CE method for rare events simulation
 - The CE method for Combinatorial Optimization
- 4 Applications
 - The max-cut problem
 - The traveling salesman problem
 - The Markovian Decision Process
 - Conclusions

We present the algorithm in the very general case:

- Let $\mathbf{X} = (X_1, \dots, X_n)$ be a random vector on \mathcal{X} .
- Let $f(\cdot; \mathbf{v})$ be a parametric family (parameter \mathbf{v}) of pdf on \mathcal{X} , with respect to some measure ν .
- Let S be some real-valued function on \mathcal{X} .

We are interested in the probability: $l = P(S(\mathbf{X}) > \gamma)$, under $f(\cdot; \mathbf{u})$.

If this probability is in the order of magnitude of 10^{-5} or less, we call the event $\{S(\mathbf{X}) > \gamma\}$ *rare event*.

We present the algorithm in the very general case:

- Let $\mathbf{X} = (X_1, \dots, X_n)$ be a random vector on \mathcal{X} .
- Let $f(\cdot; \mathbf{v})$ be a parametric family (parameter \mathbf{v}) of pdf on \mathcal{X} , with respect to some measure ν .
- Let S be some real-valued function on \mathcal{X} .

We are interested in the probability: $l = P(S(\mathbf{X}) > \gamma)$, under $f(\cdot; \mathbf{u})$.

If this probability is in the order of magnitude of 10^{-5} or less, we call the event $\{S(\mathbf{X}) > \gamma\}$ *rare event*.

Proposed methods

- 1 **MC simulation:** draw a random sample and compute $l = \frac{1}{N} \sum_{i=1}^N \mathbb{I}_{\{S(\mathbf{x}_i) > \gamma\}}$.
- 2 **Importance sampling:** We choose a function $g(\mathbf{x})$ with the same support as $f(\mathbf{x})$, that reduces the requested simulation effort.

Then, we estimate l with the following:

$$\hat{l} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}_{\{S(\mathbf{x}_i) > \gamma\}} \frac{f(\mathbf{X}_i; \mathbf{u})}{g(\mathbf{X}_i)}$$

The Kullback-Leibler distance

Again, we have the problem: how choose the function $g(\cdot)$? We know that, in order to reduce the variance as much as possible, we should have:

$$g^*(\mathbf{x}) := \frac{\mathbb{I}_{\{S(\mathbf{x}) > \gamma\}} f(\mathbf{x}; \mathbf{u})}{l},$$

but it depends from l .

The Kullback-Leibler distance

Again, we have the problem: how choose the function $g(\cdot)$? We know that, in order to reduce the variance as much as possible, we should have:

$$g^*(\mathbf{x}) := \frac{\mathbb{I}_{\{S(\mathbf{x}) > \gamma\}} f(\mathbf{x}; \mathbf{u})}{l},$$

but it depends from l .

Idea

Choose the optimal parameter \mathbf{v} (*reference parameter*), such that the distance between the densities g^* , defined as above, and a generic $f(\mathbf{x}; \mathbf{v})$ is minimal.

We use the *Kullback-Leibler* distance, or *cross-entropy* distance, defined as follows:

$$d_{KL}(g, h) = \mathbb{E} \left[\ln \frac{g(\mathbf{X})}{h(\mathbf{X})} \right] = \int g(\mathbf{x}) \ln(g(\mathbf{x})) d\mathbf{x} - \int g(\mathbf{x}) \ln(h(\mathbf{x})) d\mathbf{x}.$$

Note: It is not a real distance, because, for example, it is not symmetric.

The maximization program

Since g^* is fixed, minimizing $d_{KL}(g^*, f)$ is equivalent to find \mathbf{v} that minimizes just $-\int g^*(\mathbf{x}) \ln(f(\mathbf{x}; \mathbf{v})) d\mathbf{x}$, or that maximizes $\int g^*(\mathbf{x}) \ln(f(\mathbf{x}; \mathbf{v})) d\mathbf{x}$.

So, substituting g^* , we get the following problem:

$$\text{find } \mathbf{v} \text{ such that } \mathbb{E}_{\mathbf{u}}[\mathbb{I}_{\{S(\mathbf{X}) > \gamma\}} \ln(f(\mathbf{X}; \mathbf{v}))] \text{ is maximum}$$

With another change of variable, we get an equivalent problem in the following form:

$$\text{find } \mathbf{v} \text{ such that } \mathbb{E}_{\mathbf{w}}[\mathbb{I}_{\{S(\mathbf{X}) > \gamma\}} W(\mathbf{X}; \mathbf{u}; \mathbf{w}) \ln(f(\mathbf{X}; \mathbf{v}))] \text{ is maximum,}$$

where $W(\mathbf{x}; \mathbf{u}; \mathbf{w}) := \frac{f(\mathbf{x}; \mathbf{u})}{f(\mathbf{x}; \mathbf{w})}$ is defined *likelihood ratio*.

If we are able to draw a sample from $f(\cdot; \mathbf{w})$, we can obtain the solution solving this problem:

$$\mathbf{v} = \arg \max_{\mathbf{v}} \frac{1}{N} \sum_{i=1}^N [\mathbb{I}_{\{S(\mathbf{x}_i) > \gamma\}} W(\mathbf{X}_i; \mathbf{u}; \mathbf{w}) \ln(f(\mathbf{X}_i; \mathbf{v}))]$$

It turns out that this function is convex and differentiable in typical applications, so we can just solve the following system:

$$\sum_{i=1}^N [\mathbb{I}_{\{S(\mathbf{x}_i) > \gamma\}} W(\mathbf{X}_i; \mathbf{u}; \mathbf{w}) \nabla \ln(f(\mathbf{X}_i; \mathbf{v}))] = \mathbf{0}$$

The solution of this program can be often computed analitically, especially when the random variables belong to the natural exponential family.

If we are able to draw a sample from $f(\cdot; \mathbf{w})$, we can obtain the solution solving this problem:

$$\mathbf{v} = \arg \max_{\mathbf{v}} \frac{1}{N} \sum_{i=1}^N [\mathbb{I}_{\{S(\mathbf{x}_i) > \gamma\}} W(\mathbf{X}_i; \mathbf{u}; \mathbf{w}) \ln(f(\mathbf{X}_i; \mathbf{v}))]$$

It turns out that this function is convex and differentiable in typical applications, so we can just solve the following system:

$$\sum_{i=1}^N [\mathbb{I}_{\{S(\mathbf{x}_i) > \gamma\}} W(\mathbf{X}_i; \mathbf{u}; \mathbf{w}) \nabla \ln(f(\mathbf{X}_i; \mathbf{v}))] = \mathbf{0}$$

The solution of this program can be often computed analytically, especially when the random variables belong to the natural exponential family.

Problem

If the probability of the *Target event* $\{S(\mathbf{X}) > \gamma\}$ is very small ($\sim 10^{-5}$), most of the indicator variables would be zero, so the program above is difficult to carry out.

Multi-level algorithm

Idea

Construct a series of reference parameters $\{\mathbf{v}_t, t \geq 0\}$ and a sequence of levels $\{\gamma_t, t \geq 1\}$, and iterate both in γ_t and in \mathbf{v}_t .

Let us choose an initial value for \mathbf{v} , $\mathbf{v}_0 = \mathbf{u}$, and a parameter ρ not very small ($\sim 10^{-2}$); now let us fix γ_1 such that $l_1 \geq \rho$, i.e. $\mathbb{E}_{\mathbf{v}_0}[\mathbb{I}_{\{S(\mathbf{x}) > \gamma_1\}}] \geq \rho$. $\gamma_1 < \gamma!$

Now we find \mathbf{v}_1 solving the maximization program with l_1 . Then, we compute γ_2 such that $\mathbb{E}_{\mathbf{v}_1}[\mathbb{I}_{\{S(\mathbf{x}) > \gamma_2\}}] \geq \rho$, then we compute l_2 , and so on.

Multi-level algorithm

Idea

Construct a series of reference parameters $\{\mathbf{v}_t, t \geq 0\}$ and a sequence of levels $\{\gamma_t, t \geq 1\}$, and iterate both in γ_t and in \mathbf{v}_t .

Let us choose an initial value for \mathbf{v} , $\mathbf{v}_0 = \mathbf{u}$, and a parameter ρ not very small ($\sim 10^{-2}$); now let us fix γ_1 such that $l_1 \geq \rho$, i.e. $\mathbb{E}_{\mathbf{v}_0}[\mathbb{I}_{\{S(\mathbf{x}) > \gamma_1\}}] \geq \rho$. $\gamma_1 < \gamma!$

Now we find \mathbf{v}_1 solving the maximization program with l_1 . Then, we compute γ_2 such that $\mathbb{E}_{\mathbf{v}_1}[\mathbb{I}_{\{S(\mathbf{x}) > \gamma_2\}}] \geq \rho$, then we compute l_2 , and so on.

Algorithm:

- 1 draw a sample $(\mathbf{X}_1, \dots, \mathbf{X}_N)$ from $f(\cdot; \mathbf{v}_{t-1})$
- 2 compute the performance indices $S(\mathbf{X}_i)$, and calculate the $(1 - \rho)$ th sample quartile, i.e. $\gamma_t = S_{(\lceil(1-\rho)N\rceil)}$
- 3 compute $\mathbf{v}_t = \arg \max_{\mathbf{v}} \frac{1}{N} \sum_{i=1}^N [\mathbb{I}_{\{S(\mathbf{x}_i) > \gamma_t\}} W(\mathbf{X}_i; \mathbf{u}; \mathbf{v}_{t-1}) \ln(f(\mathbf{X}_i; \mathbf{v}))]$

Complete algorithm (resume):

- 1 Define $\mathbf{v}_0 = \mathbf{u}$. Set $t = 1$
- 2 While $\hat{\gamma}_t \neq \gamma$
 - 1 Generate a random sample $(\mathbf{X}_1, \dots, \mathbf{X}_N)$ from $f(\cdot; \mathbf{v}_{t-1})$
 - 2 Calculate the performances $S(\mathbf{X}_i)$ and order them \nearrow .
 - 3 Let $\hat{\gamma}_t := S_{(\lceil(1-\rho)N\rceil)}$, i.e. the $(1 - \rho)$ th quartile, provided that is less than γ . Otherwise, $\hat{\gamma}_t = \gamma$.
 - 4 Compute \mathbf{v}_t as follows (with the same sample $(\mathbf{X}_1, \dots, \mathbf{X}_N)$):

$$\mathbf{v}_t = \arg \max_{\mathbf{v}} \frac{1}{N} \sum_{i=1}^N \mathbb{I}_{\{S(\mathbf{x}_i) > \hat{\gamma}_t\}} W(\mathbf{X}_i; \mathbf{u}; \mathbf{v}_{t-1}) \ln(f(\mathbf{X}_i; \mathbf{v})) \quad (1)$$

- 3 Estimate the rare-event probability l as:

$$\hat{l} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}_{\{S(\mathbf{x}_i) > \hat{\gamma}\}} W(\mathbf{X}_i; \mathbf{u}; \hat{\mathbf{v}}_T) \quad (2)$$

...from the first example...

We recall that:

$$f(\mathbf{x}; \mathbf{v}) = \prod_{j=1}^5 \frac{1}{v_j} \exp \left\{ - \sum_{j=1}^5 \frac{x_j}{v_j} \right\} \Rightarrow \frac{\partial}{\partial v_j} \ln(f(\mathbf{x}; \mathbf{v})) = \frac{x_j}{v_j^2} - \frac{1}{v_j}$$

So, when we want to solve the maximization problem, we get a system of equations like this one:

$$\sum_{i=1}^N \left[\mathbb{I}_{\{S(\mathbf{x}_i) > \gamma\}} W(\mathbf{X}_i; \mathbf{u}; \mathbf{w}) \left(\frac{X_{ij}}{v_j^2} - \frac{1}{v_j} \right) \right] = \mathbf{0},$$

and finally these updating equations:

$$v_j = \frac{\sum_{i=1}^N \mathbb{I}_{\{S(\mathbf{x}_i) > \gamma\}} W(\mathbf{X}_i; \mathbf{u}; \mathbf{w}) X_{ij}}{\sum_{i=1}^N \mathbb{I}_{\{S(\mathbf{x}_i) > \gamma_t\}} W(\mathbf{X}_i; \mathbf{u}; \mathbf{w})}$$

Last remark on this algorithm

If we know the distribution of $S(\mathbf{X})$ and of the others quantities involved, we can compute the expected values and the exact quantiles instead of the sample means and the empirical quantiles.

So we can replace:

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

with:

$$\gamma_t := \max\{s : P_{\mathbf{v}_{t-1}}(S(\mathbf{X}) \geq s) \geq \rho\},$$

and

$$\mathbf{v}_t = \arg \max_{\mathbf{v}} \frac{1}{N} \sum_{i=1}^N [\mathbb{I}_{\{S(\mathbf{x}_i) > \hat{\gamma}_t\}} W(\mathbf{X}_i; \mathbf{u}; \mathbf{v}_{t-1}) \ln(f(\mathbf{X}_i; \mathbf{v}))]$$

with:

$$\mathbf{v}_t = \arg \max_{\mathbf{v}} \mathbb{E}_{\mathbf{v}_{t-1}} [\mathbb{I}_{\{S(\mathbf{x}) > \gamma_t\}} W(\mathbf{X}; \mathbf{u}; \mathbf{v}_{t-1}) \ln(f(\mathbf{X}; \mathbf{v}))]$$

We call it *deterministic* version of the CE algorithm.

The CE method for Combinatorial Optimization

- \mathcal{X} : finite set of states.
- S : 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:

$$\max_{\mathbf{x} \in \mathcal{X}} S(\mathbf{x}) = \gamma^* = S(\mathbf{x}^*). \quad (3)$$

The CE method for Combinatorial Optimization

- \mathcal{X} : finite set of states.
- S : 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:

$$\max_{\mathbf{x} \in \mathcal{X}} S(\mathbf{x}) = \gamma^* = S(\mathbf{x}^*). \quad (3)$$

In order to read this problem in terms of CE problem, we have to define an *estimation problem* associated with this maximization problem. So, we define:

- a collection of indicator functions $\{\mathbb{I}_{\{S(\mathbf{x}) \geq \gamma\}}\}$ for different levels γ
- a family of parametrized pdfs on \mathcal{X} , $f(\mathbf{x}; \mathbf{u})$.

Now we can associate (3) with the following problem (Associated Stochastic Program):

$$\text{find } l(\gamma) = P_{\mathbf{u}}(S(\mathbf{X}) \geq \gamma) = \mathbb{E}[\mathbb{I}_{\{S(\mathbf{x}) \geq \gamma\}}] = \sum_{\mathbf{x}} [\mathbb{I}_{\{S(\mathbf{x}) \geq \gamma\}} f(\mathbf{x}; \mathbf{u})] \quad (4)$$

From a maximization problem to an estimation problem

How (4) is associated with (3)?

Suppose that $\gamma = \gamma^*$ and $f(\mathbf{x}; \mathbf{u})$ is the uniform distribution over \mathcal{X} . Then, $l(\gamma^*)$ is a very small number, because the indicator function is one only when $\mathbf{X} = \mathbf{x}^*$, so the sum contains just one term ($\frac{1}{|\mathcal{X}|}$).

So, we can think about using the previous algorithm in order to estimate this probability. In particular, we can choose as reference parameter \mathbf{v}^* , where:

$$\mathbf{v}^* = \arg \max_{\mathbf{v}} \frac{1}{N} \sum_{i=1}^N [\mathbb{I}_{\{S(\mathbf{x}_i) \geq \gamma\}} \ln f(\mathbf{X}_i; \mathbf{v})], \quad (5)$$

where \mathbf{X}_i are samples from $f(\mathbf{x}; \mathbf{u})$; then we estimate the requested probability with (2).

The **problem** now is very similar to that one we discussed before: how choosing γ and \mathbf{u} such that $P_{\mathbf{u}}(S(\mathbf{X}) \geq \gamma)$ is not too small.

Idea

The idea now is to perform a two-phase multilevel approach, *in which we simultaneously construct a sequence of levels $\gamma_1, \dots, \gamma_T$ and vectors $\mathbf{v}_1, \dots, \mathbf{v}_T$ such that γ_T is close to the optimal γ^* and \mathbf{v}_T is such that the corresponding density assigns high probability mass to the collection of states that give a high performance.*

What we are going to do is the following (from *Rubinstein, '99*):

- break down the "hard" problem, of estimating a very small quantity, the partition function, into a sequence of associated "simple" continuous and smooth optimization problems
- solve this sequence of problems based on the Kullback-Leibner distance
- take the mode of the Importance Sampling pdf, which converges to the unit mass distribution concentrated at the point x^* , as the estimate of the optimal solution x^* .

Complete algorithm for Optimization problems:

- 1 Define $\mathbf{v}_0 = \mathbf{u}$. Set $t = 1$
- 2 While stop criterion
 - 1 Generate a random sample $(\mathbf{X}_1, \dots, \mathbf{X}_N)$ from $f(\cdot; \mathbf{v}_{t-1})$
 - 2 Calculate the performances $S(\mathbf{X}_i)$ and order them \nearrow .
 - 3 Let $\hat{\gamma}_t := S_{(\lceil (1-\rho)N \rceil)}$, i.e. the $(1 - \rho)$ th quartile.
 - 4 Compute \mathbf{v}_t as follows (with the same sample $(\mathbf{X}_1, \dots, \mathbf{X}_N)$):

$$\mathbf{v}_t = \arg \max_{\mathbf{v}} \frac{1}{N} \sum_{i=1}^N [\mathbb{I}_{\{S(\mathbf{X}_i) > \hat{\gamma}_t\}} \cdot 1 \cdot \ln(f(\mathbf{X}_i; \mathbf{v}))] \quad (6)$$

- 3 Set $\gamma^* = \gamma_T$ and $\mathbf{x}^* = \max_{\mathbf{x}} f(\mathbf{x}; \mathbf{v}_T)$

A possible stop criterion is choosing d , and stopping the procedure when $\gamma_t = \gamma_{t-1} = \dots = \gamma_{t-d}$.

Some remarks (1)

Main differences between RE and CO algorithms::

- **The role of \mathbf{u} (1)**: In the RE algorithm, we use \mathbf{u} in every step of the algorithm for the computation of $W(\mathbf{X}_i, \mathbf{u}, \mathbf{v}_{t-1})$, while in the CO algorithm we forget about it after the first iteration.
- **The role of \mathbf{u} (2)**: In the RE algorithm the Associated Stochastic Problem is unique ($P_{\mathbf{u}}(S(\mathbf{X}) \geq \gamma_t)$), while in the CO algorithm it is re-defined after each iteration ($P_{\mathbf{v}_{t-1}}(S(\mathbf{X}) \geq \gamma_t)$).
- **The role of W** : in the CO algorithm $W = 1$, because \mathbf{v}_{t-1} is used directly in the ASP problem.

Some remarks (1)

Main differences between RE and CO algorithms::

- **The role of \mathbf{u} (1)**: In the RE algorithm, we use \mathbf{u} in every step of the algorithm for the computation of $W(\mathbf{X}_i, \mathbf{u}, \mathbf{v}_{t-1})$, while in the CO algorithm we forget about it after the first iteration.
- **The role of \mathbf{u} (2)**: In the RE algorithm the Associated Stochastic Problem is unique ($P_{\mathbf{u}}(S(\mathbf{X}) \geq \gamma_t)$), while in the CO algorithm it is re-defined after each iteration ($P_{\mathbf{v}_{t-1}}(S(\mathbf{X}) \geq \gamma_t)$).
- **The role of W** : in the CO algorithm $W = 1$, because \mathbf{v}_{t-1} is used directly in the ASP problem.

Applications of the CO algorithm: we can apply the CE-algorithm in any optimization problem, but we have to state carefully these ingredients:

- We need to specify the family of pdf ($f(\cdot; \mathbf{v})$) that allow us to generate the samples.
- We need to calculate the updating rules for the parameters, i.e. the way in which we compute γ_t and \mathbf{v}_t .

Some remarks (2)

ML Estimation: The updating of the reference parameter \mathbf{v}^* could be also considered in terms of MLE.

We know that if we are searching the MLE of a parameter \mathbf{v} on the basis of a sample $(\mathbf{X}_1, \dots, \mathbf{X}_N)$ from $f(\cdot; \mathbf{v})$, we could express it as:

$$\hat{\mathbf{v}} = \arg \max_{\mathbf{v}} \sum_{i=1}^N \ln f(\mathbf{X}_i; \mathbf{v})$$

Now, if we compare this expression with the expressions (5) and (6), we will see that the only difference is the presence of the indicator function. We can rewrite (6) in this way:

$$\hat{\mathbf{v}}_t = \arg \max_{\mathbf{v}} \sum_{\mathbf{X}_i: S(\mathbf{X}_i) \geq \gamma_t} \ln f(\mathbf{X}_i; \mathbf{v}),$$

so we can say that $\hat{\mathbf{v}}_t$ 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 γ_t .

Some remarks (3)

Choice of parameters: The algorithm computes autonomously the parameters needed, except from ρ and N (sample size). The paper suggests to choose:

- In a **SNN-type** problem: $N = cn$, where c is a constant and n is the number of nodes.
- In a **SEN-type** problem: $N = cn^2$, where c is a constant and n^2 is the number of edges.

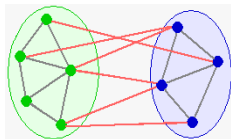
The paper suggests to take ρ around 0.01, provided n is reasonably large ($n \geq 100$), and larger if $n < 100$.

- 1 Introduction
- 2 Two examples
 - Rare event simulation
 - Combinatorial Optimization
- 3 The main algorithm
 - The CE method for rare events simulation
 - The CE method for Combinatorial Optimization
- 4 Applications**
 - The max-cut problem
 - The traveling salesman problem
 - The Markovian Decision Process
 - Conclusions

The max-cut problem

The problem is a SNN with these features:

- Weighted graph $G = \{V, E\}$; nodes $V = \{1, \dots, n\}$, edges E .
- Every edge has a non negative fixed weight.



Aim:

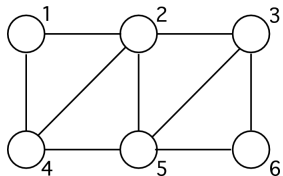
Find a partition of the graph in two subsets such that the sum of the weights of the edges going from one subset to the other is maximized.

We will call this partition $\{V_1, V_2\}$, *cut*.

We define **cost** of a cut, the sum of all the weights across the cut.

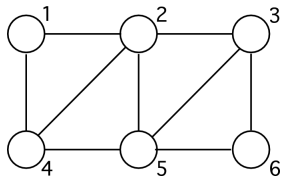
Problem: The max-cut problem is an NP-hard problem

The max-cut problem

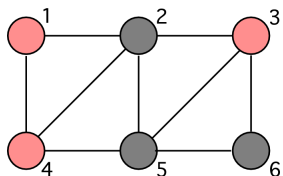


$$C = \begin{pmatrix} 0 & c_{12} & c_{13} & 0 & 0 & 0 \\ c_{21} & 0 & c_{23} & c_{24} & 0 & 0 \\ c_{31} & c_{32} & 0 & c_{34} & c_{35} & 0 \\ 0 & c_{42} & c_{43} & 0 & c_{45} & c_{46} \\ 0 & 0 & c_{53} & c_{54} & 0 & c_{56} \\ 0 & 0 & 0 & c_{64} & c_{65} & 0 \end{pmatrix}$$

The max-cut problem



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



If we select the cut $\{\{1, 3, 4\}, \{2, 5, 6\}\}$, we have the following cost:

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

Cut vector: $\{1, 0, 1, 1, 0, 0\}$

The CE method applied to the max-cut problem

- \mathcal{X} : set of all possible cut vectors $\mathbf{x} = \{1, x_2, \dots, x_n\}$
- $S(\mathbf{x})$: corresponding cost of the cut.

We can read the problem in the framework presented in the previous part, and let the cut vectors be realizations of independent Bernoulli r.v.

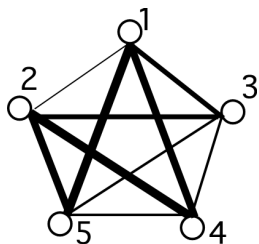
$$\Rightarrow X_2, \dots, X_n \sim \text{Be}(p_2), \dots, \text{Be}(p_n).$$

We already have an updating expression for the parameters:

$$\hat{p}_{t,j} = \frac{\sum_{i=1}^N \mathbb{I}_{\{S(\mathbf{x}_i) > \hat{\gamma}_t\}} \mathbb{I}_{\{X_{ij}=1\}}}{\sum_{i=1}^N \mathbb{I}_{\{S(\mathbf{x}_i) > \hat{\gamma}_t\}}} \quad j = 2, \dots, n$$

Remark: We can extend the procedure to the case in which the node set V is partitioned into $r > 2$ subsets. In this case one can follow the basic steps of the algorithm for CO problems, using independent r point distributions instead of independent Bernoulli distributions.

Example 1 (simple)



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

In this case the optimal cut vector is $\mathbf{x}^* = (1, 1, 0, 0, 0)$ with $S(\mathbf{x}^*) = \gamma^* = 28$.

We solve this max-cut problem using the **deterministic** version of the algorithm.

We start from $\mathbf{p}_0 = (1, 1/2, 1/2, 1/2, 1/2)$ and $\rho = 0.1$.

Example 1 (simple)

- 1 We have to fix $\gamma_1 = \max\{\gamma : P_{p_0}(S(\mathbf{X}) \geq \gamma) \geq 0.1\}$. We know all the possible cuts, so we can compute all the possible values of $S(\mathbf{x})$ and compute directly this quantile.
- 2 $S(\mathbf{x}) = \{0, 10, 15, 18, 19, 20, 21, 26, 28\}$ with probabilities:
 $\{1/16, 1/16, 1/4, 1/8, 1/8, 1/8, 1/8, 1/16, 1/16\} \Rightarrow \gamma_1 = 26$.

Example 1 (simple)

- 1 We have to fix $\gamma_1 = \max\{\gamma : P_{p_0}(S(\mathbf{X}) \geq \gamma) \geq 0.1\}$. We know all the possible cuts, so we can compute all the possible values of $S(\mathbf{x})$ and compute directly this quantile.
- 2 $S(\mathbf{x}) = \{0, 10, 15, 18, 19, 20, 21, 26, 28\}$ with probabilities:
 $\{1/16, 1/16, 1/4, 1/8, 1/8, 1/8, 1/8, 1/16, 1/16\} \Rightarrow \gamma_1 = 26$.
- 3 Now we need to update \mathbf{p} and compute \mathbf{p}_1 as:

$$\hat{p}_{1,j} = \frac{\mathbb{E}_{p_0}[\mathbb{I}_{\{S(\mathbf{x}) \geq \gamma_1\}}] X_j}{\mathbb{E}_{p_0}[\mathbb{I}_{\{S(\mathbf{x}) \geq \gamma_1\}}]} \quad j = 2, \dots, 5$$

- 4 Since only $\mathbf{x}_1 = \{1, 1, 1, 0, 0\}$ and $\mathbf{x}_2 = \{1, 1, 0, 0, 0\}$ have $S(\mathbf{x}) \geq \gamma_1$ and both have probability $1/16$, we get:

$$p_{1,1} = p_{1,2} = \frac{2/16}{2/16} = 1 \quad p_{1,3} = \frac{1/16}{2/16} = \frac{1}{2} \quad p_{1,4} = p_{1,5} = \frac{0}{2/16} = 0$$

Example 1 (simple)

- 1 We have to fix $\gamma_1 = \max\{\gamma : P_{p_0}(S(\mathbf{X}) \geq \gamma) \geq 0.1\}$. We know all the possible cuts, so we can compute all the possible values of $S(\mathbf{x})$ and compute directly this quantile.
- 2 $S(\mathbf{x}) = \{0, 10, 15, 18, 19, 20, 21, 26, 28\}$ with probabilities:
 $\{1/16, 1/16, 1/4, 1/8, 1/8, 1/8, 1/8, 1/16, 1/16\} \Rightarrow \gamma_1 = 26$.

- 3 Now we need to update \mathbf{p} and compute \mathbf{p}_1 as:

$$\hat{p}_{1,j} = \frac{\mathbb{E}_{p_0}[\mathbb{I}_{\{S(\mathbf{x}) \geq \gamma_1\}}] X_j}{\mathbb{E}_{p_0}[\mathbb{I}_{\{S(\mathbf{x}) \geq \gamma_1\}}]} \quad j = 2, \dots, 5$$

- 4 Since only $\mathbf{x}_1 = \{1, 1, 1, 0, 0\}$ and $\mathbf{x}_2 = \{1, 1, 0, 0, 0\}$ have $S(\mathbf{x}) \geq \gamma_1$ and both have probability $1/16$, we get:

$$p_{1,1} = p_{1,2} = \frac{2/16}{2/16} = 1 \quad p_{1,3} = \frac{1/16}{2/16} = \frac{1}{2} \quad p_{1,4} = p_{1,5} = \frac{0}{2/16} = 0$$

- 5 Under \mathbf{p}_1 , $\mathbf{X} \in \{(1, 1, 0, 0, 0), (1, 1, 1, 0, 0)\}$. So $S(\mathbf{X}) = \{26, 28\}$ with probabilities: $\{1/2, 1/2\}$, $\Rightarrow \gamma_2 = 28 \Rightarrow \mathbf{p}_2 = (1, 1, 0, 0, 0)$.

Example 2 (complex)

We consider an artificial n -dimension problem with this block cost matrix:

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

where:

- $\{Z_{11}\}_{i < j} \sim U(a, b)$ for each $i, j \in \{1, \dots, m\}$, $m \in 1, \dots, n$
- $\{Z_{22}\}_{i < j} \sim U(a, b)$ for each $i, j \in \{n - m, \dots, n\}$, $m \in 1, \dots, n$
- $\{Z_{11}\}_{ii} = \{Z_{22}\}_{ii} = 0$
- $\{B_{12}\}_{ij} = \{B_{21}\}_{ij} = c$ for each i, j .

Note: If $c > \frac{b(n-m)}{m}$, then the optimal cut is $\{V_1^*, V_2^*\}$, where $V_1^* = \{1, \dots, m\}$ and $V_2^* = \{m + 1, \dots, n\}$. The optimal value is $\gamma^* = cm(n - m)$.

Example 2 (complex)

We tried to simulate such example and compare the simulation with the results in the paper.

We have the following situation:

- Model parameters: $n = 400$, $m = 200$.
- Algorithm parameters: $\rho = 0.1$, $N = 1000$, $\mathbf{p}_0 = (1/2, \dots, 1/2)$.

At each iteration we perform N simulations drawn from $f(\cdot; \mathbf{p}_{t-1})$ and compute the cost of the cut for each simulation. Then we apply the CE algorithm for Combinatorial Optimization problems.

Example 2 (complex)

We tried to simulate such example and compare the simulation with the results in the paper.

We have the following situation:

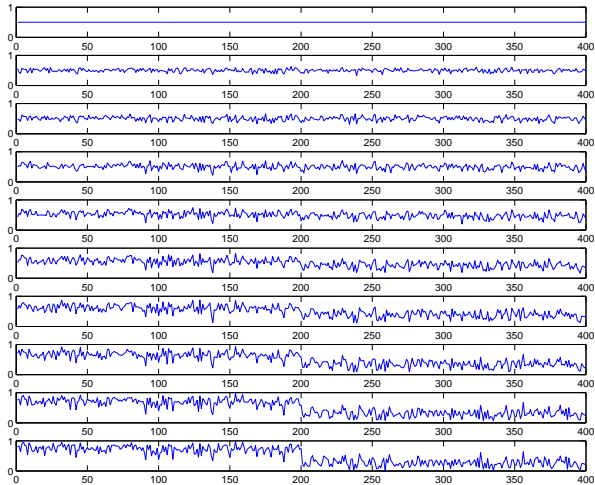
- Model parameters: $n = 400$, $m = 200$.
- Algorithm parameters: $\rho = 0.1$, $N = 1000$, $\mathbf{p}_0 = (1/2, \dots, 1/2)$.

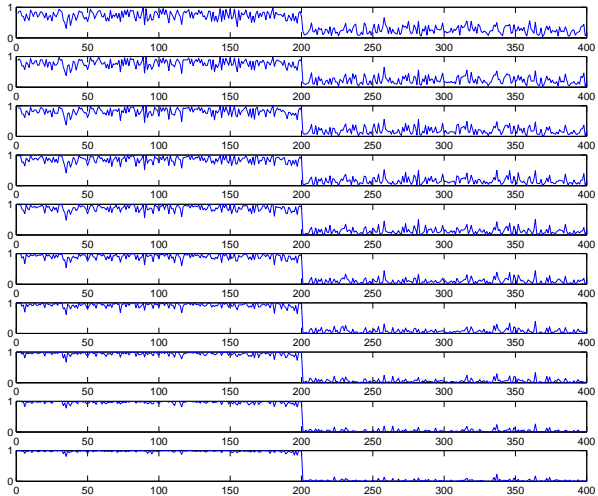
At each iteration we perform N simulations drawn from $f(\cdot; \mathbf{p}_{t-1})$ and compute the cost of the cut for each simulation. Then we apply the CE algorithm for Combinatorial Optimization problems.

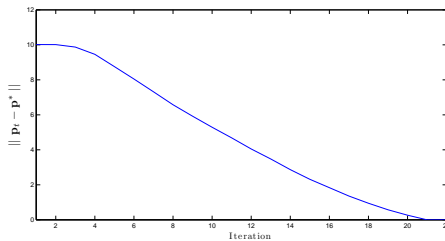
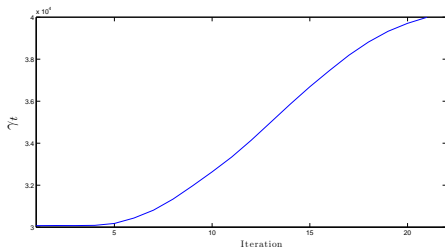
```
for i=1:N
    sampl(i,:)=binornd(1,pt);
    for j=1:n
        for k=(j+1):n
            if sampl(i,j)~=sampl(i,k)
                lugh(i)=lugh(i)+C(j,k);
            end
        end
    end
end
end
gammat=quantile(lugh,(1-rho));
pt=(+(lugh>gammat)*(sampl==1))./(sum(lugh>gammat));
```

Same results as in the paper:

- < 5 seconds per iteration
- convergence in 20 iterations
- optimal reference vector $\mathbf{p}^* = (1, \dots, 1, 0, \dots, 0)$







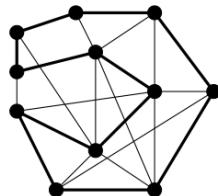
- $(1 - \rho)$ -quantile of the performances (γ_t)

- $\|p_t - p^*\| = \sqrt{\sum_i (p_{t,i} - p_i^*)^2}$

The traveling salesman problem

The problem:

- Weighted graph $G = \{V, E\}$; nodes $V = \{1, \dots, n\}$, edges E .
- Nodes=cities, edges=roads.
- Each edge has a weight c_{ij} , representing the length of the road



Aim:

Find the shortest tour that visits all the cities exactly once.

We will consider complete graphs (without loss of generality)

We can represent each tour $\mathbf{x} = (x_1, \dots, x_n)$ via a permutation of $(1, \dots, n)$

Problem: The traveling-salesman problem is an NP-hard problem, too.

The traveling salesman problem

If we define $S(\mathbf{x})$ the total length of tour \mathbf{x} , the aim of our problem becomes:

$$\min_{\mathbf{x} \in \mathcal{X}} S(\mathbf{x}) = \min_{\mathbf{x} \in \mathcal{X}} \left\{ \sum_{i=1}^{n-1} c_{x_i, x_{i+1}} + c_{x_n, x_1} \right\}$$

Now we need to specify how to generate the random tours, and how to update the parameters at each iteration.

If we define $\tilde{\mathcal{X}} = \{(x_1, \dots, x_n) : x_1 = 1, x_j \in \{1, \dots, n\}, j = 2, \dots, n\}$, we can generate a path \mathbf{X} on $\tilde{\mathcal{X}}$ with an n -step Markov chain.

We can write the log-density of \mathbf{X} in this way:

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

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 traveling salesman problem

Introducing the constraint $\sum_j p_{ij} = 1$, and differentiating the corresponding maximization problem, we get the following updating rule for the elements of the matrix P :

$$p_{ij} = \frac{\sum_{k=1}^N \left[\mathbb{I}_{\{\tilde{S}(\mathbf{x}_k) \leq \gamma\}} \sum_{r=1}^n \mathbb{I}_{\{\mathbf{x}_k \in \tilde{\mathcal{X}}_{ij}(r)\}} \right]}{\sum_{k=1}^N \left[\mathbb{I}_{\{\tilde{S}(\mathbf{x}_k) \leq \gamma\}} \sum_{r=1}^n \mathbb{I}_{\{\mathbf{x}_k \in \tilde{\mathcal{X}}_i(r)\}} \right]}$$

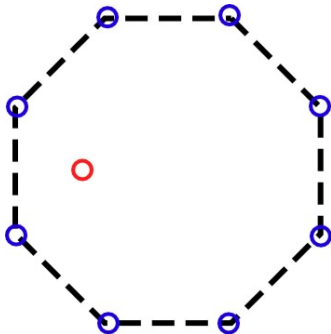
Interpretation: To update p_{ij} we simply take the fraction of times that the transitions from i to j occurred, taking into account only those paths that have a total length less than or equal to γ .

Note (1) : We use $\tilde{S}(\mathbf{x})$ in stead of $S(\mathbf{x})$, because it could happen that $\mathbf{x} \in \tilde{\mathcal{X}}$, but $\mathbf{x} \notin \mathcal{X}$. So, we define $\tilde{S}(\mathbf{x}) = S(\mathbf{x})$ if $\mathbf{x} \in \mathcal{X}$ and $\tilde{S}(\mathbf{x}) = \infty$ otherwise.

Note (2) : 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 ∞ . We can modify the algorithm in order to avoid the generation of such paths.

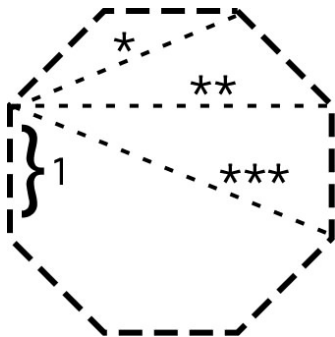
The traveling salesman problem: first example

- We can build a simple example where we have 8 cities disposed in the following way:
- We add a one more city in this position:



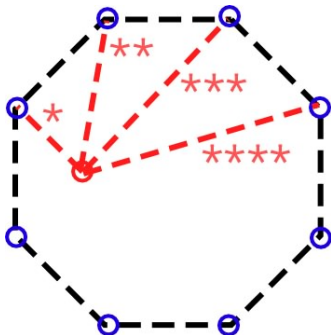
The traveling salesman problem: first example

- We can build a simple example where we have 8 cities disposed in the following way:
- We add a one more city in this position:
- We can compute the distances for the original cities...



The traveling salesman problem: first example

- We can build a simple example where we have 8 cities disposed in the following way:
- We add a one more city in this position:
- ...and for the new one



The traveling salesman problem: first example

In this simple case the matrix of the distance has this form:

$$C = \begin{pmatrix} 0 & 1 & * & ** & *** & ** & * & 1 & * \\ 1 & 0 & 1 & * & ** & *** & ** & * & * \\ * & 1 & 0 & 1 & * & ** & *** & ** & ** \\ ** & * & 1 & 0 & 1 & * & ** & *** & *** \\ *** & ** & * & 1 & 0 & 1 & * & ** & **** \\ ** & *** & ** & * & 1 & 0 & 1 & * & **** \\ * & ** & *** & ** & * & 1 & 0 & 1 & *** \\ 1 & * & ** & *** & ** & * & 1 & 0 & ** \\ * & * & ** & *** & **** & **** & *** & ** & 0 \end{pmatrix}$$

The traveling salesman problem: first example

In this simple case the matrix of the distance has this form:

$$C = \begin{pmatrix} 0 & 1 & * & ** & *** & ** & * & 1 & * \\ 1 & 0 & 1 & * & ** & *** & ** & * & * \\ * & 1 & 0 & 1 & * & ** & *** & ** & ** \\ ** & * & 1 & 0 & 1 & * & ** & *** & *** \\ *** & ** & * & 1 & 0 & 1 & * & ** & **** \\ ** & *** & ** & * & 1 & 0 & 1 & * & **** \\ * & ** & *** & ** & * & 1 & 0 & 1 & *** \\ 1 & * & ** & *** & ** & * & 1 & 0 & ** \\ * & * & ** & *** & **** & **** & *** & ** & 0 \end{pmatrix}$$

With this numeration, the optimal path is, obviously, $\{1, 9, 2, 3, 4, 5, 6, 7, 8\}$ or $\{1, 8, 7, 6, 5, 4, 3, 2, 9\}$.

The traveling salesman problem: first example

In this simple case the matrix of the distance has this form:

$$C = \begin{pmatrix} 0 & 1 & * & ** & *** & ** & * & 1 & * \\ 1 & 0 & 1 & * & ** & *** & ** & * & * \\ * & 1 & 0 & 1 & * & ** & *** & ** & ** \\ ** & * & 1 & 0 & 1 & * & ** & *** & *** \\ *** & ** & * & 1 & 0 & 1 & * & ** & **** \\ ** & *** & ** & * & 1 & 0 & 1 & * & **** \\ * & ** & *** & ** & * & 1 & 0 & 1 & *** \\ 1 & * & ** & *** & ** & * & 1 & 0 & ** \\ * & * & ** & *** & **** & **** & *** & ** & 0 \end{pmatrix}$$

With this numeration, the optimal path is, obviously, $\{1, 9, 2, 3, 4, 5, 6, 7, 8\}$ or $\{1, 8, 7, 6, 5, 4, 3, 2, 9\}$.

We simulated this example with the function `tsp.m`; we get the optimal path in 8 iterations, with an optimal length of 8.4142 ($7 + 2\sqrt{2}/2!$).

The traveling salesman problem: first example

In this case we start with a uniform P matrix:

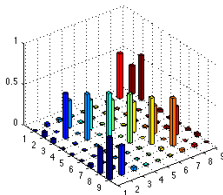
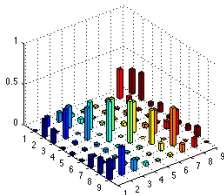
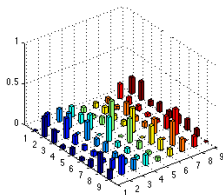
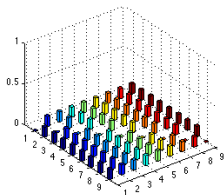
$$P = \begin{pmatrix} 0 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 \\ 0.1250 & 0 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 \\ 0.1250 & 0.1250 & 0 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 \\ 0.1250 & 0.1250 & 0.1250 & 0 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 \\ 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0 & 0.1250 & 0.1250 & 0.1250 & 0.1250 \\ 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0 & 0.1250 & 0.1250 & 0.1250 \\ 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0 & 0.1250 & 0.1250 \\ 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0 & 0.1250 \\ 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0 \end{pmatrix}$$

and we get at the final iteration a band P matrix of this kind:

$$P = \begin{pmatrix} 0 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.3560 & 0.6439 \\ 0.0000 & 0 & 0.6439 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.3560 \\ 0.0000 & 0.3560 & 0 & 0.6440 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.3560 & 0 & 0.6440 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.3560 & 0 & 0.6440 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.3560 & 0 & 0.6439 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.3560 & 0 & 0.6439 & 0.0000 \\ 0.6439 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.3560 & 0 & 0.0000 \\ 0.3560 & 0.6440 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0 \end{pmatrix}$$

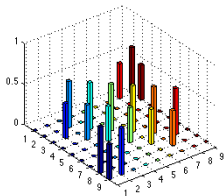
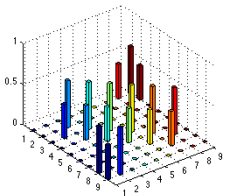
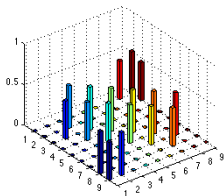
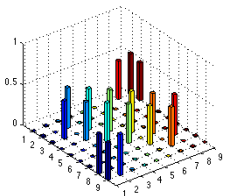
The traveling salesman problem: first example

Bar plot of the P matrices: iterations 1 – 4



The traveling salesman problem: first example

Bar plot of the P matrices: iterations 5 – 8



The traveling salesman problem: second example

In the paper is discussed an example concerning the dataset ft53:

- distances between 53 cities
- dataset used to test maximization algorithms
- best known solution: $\gamma^* = 6905$
- the authors run the CE algorithm with $\rho = 0.01$ and $N = 28090$ and they get as best solution $\gamma_T = 7008$ in 31 iterations, with a computational time of 6 minutes.

The traveling salesman problem: second example

In the paper is discussed an example concerning the dataset ft53:

- distances between 53 cities
- dataset used to test maximization algorithms
- best known solution: $\gamma^* = 6905$
- the authors run the CE algorithm with $\rho = 0.01$ and $N = 28090$ and they get as best solution $\gamma_T = 7008$ in 31 iterations, with a computational time of 6 minutes.

We tried to replicate their results, but we did not managed to obtain their results, because of a larger computational time. We reached as best result $\gamma_T = 7120$ in 35 iterations, with a computational time of 120 minutes. ($\rho = 0.03$, $N = 10000$)

Note: The results are good even with relatively high values of ρ (0.05) and small values of N (1000) (and a shorter computational time!) :

iteration 1 : $\gamma = 24123$ iteration 5 : $\gamma = 17844$
iteration 10 : $\gamma = 13997$ iteration 20 : $\gamma = 10493$
iteration 30 : $\gamma = 8789$ iteration 39 : $\gamma = 8038$

The Markovian decision process

The Markovian decision process (MDP) model is broadly used in many fields, such as artificial intelligence, machine learning, operation research...

Definition:

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

- $\mathcal{Z} = \{1, \dots, n\}$ is a finite set of states.
- $\mathcal{A} = \{1, \dots, m\}$ is the set of possible actions by the decision maker.
- \mathcal{P} is the transition probability matrix with elements $\mathcal{P}(z'|z, a)$ presenting the transition probability from state z to state z' , when action a is chosen.
- $r(z, a)$ 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 $\mathcal{P}(z'|z_k, a_k)$.

A **policy** π determines, for each history $H_k = \{z_1, a_1, \dots, a_{k-1}, z_k\}$ of states and actions, the probability distribution of the decision maker's action at time k . A policy is called **Markov** if each action is deterministic, and depends only on the current state z_k .

Finally, a Markov policy is called **stationary** if it does not depend on the time k .

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 $\mathcal{P}(z'|z_k, a_k)$.

A **policy** π determines, for each history $H_k = \{z_1, a_1, \dots, a_{k-1}, z_k\}$ of states and actions, the probability distribution of the decision maker's action at time k . A policy is called **Markov** if each action is deterministic, and depends only on the current state z_k .

Finally, a Markov policy is called **stationary** if it does not depend on the time k .

Aim: Maximizing the total reward:

$$V(\pi, z_0) = \mathbb{E}_\pi \sum_{k=0}^{\tau-1} r_k, \quad (7)$$

starting from some fixed state z_0 and finishing at τ . Here \mathbb{E}_π denotes the expectation with respect to some probability measure induced by the policy π .

Note: We will restrict attention to stochastic shortest path MDP, where it is assumed that the process starts from a specific initial state $z_0 = z_{start}$, and that there is a absorbing state z_{fin} with zero reward.

The MDP problem in the CE framework

We can represent each stationary policy as a vector $\mathbf{x} = (x_1, \dots, x_n)$, with $x_i \in \{1, \dots, m\}$ being the action taken when visiting state i . So, we can rewrite (7) as

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

where Z_0, Z_1, \dots are the states visited, and A_0, A_1, \dots are the actions taken.

Idea

Combine the random policy generation and the random trajectory generation in the following way: at each stage of the CE algorithm:

- 1 We generate N random trajectories $(Z_0, A_0, Z_1, A_1, \dots, Z_\tau, A_\tau)$ using an auxiliary policy matrix P ($n \times m$) and the transition matrix \mathcal{P} , and we compute the cost of each trajectory: $\hat{S}(\mathbf{X}) = \sum_{j=0}^{\tau-1} r(Z_j, A_j)$.
- 2 We update of the parameters of the policy matrix $\{p_{za}\}$ on the basis of the data collected at the first phase, following the CE algorithm.

The MDP algorithm

(1) We initialize the policy matrix P as uniform matrix

(2) While stop criterion

① for $i=1:N$

① We start from the given initial state $Z_0 = z_{start}$, $k = 0$.

② While $z_k \neq z_{fin}$

① We generate an action A_k according to the Z_k th row of P

② We calculate the reward $r_k = r(Z_k, A_k)$

③ We generate a new state Z_{k+1} according to $\mathcal{P}(\cdot|Z_k, A_k)$ and set $k = k + 1$.

③ We compute the score of the trajectory $\mathbf{X} = \{z_{start}, A_1, \dots, z_{fin}\}$

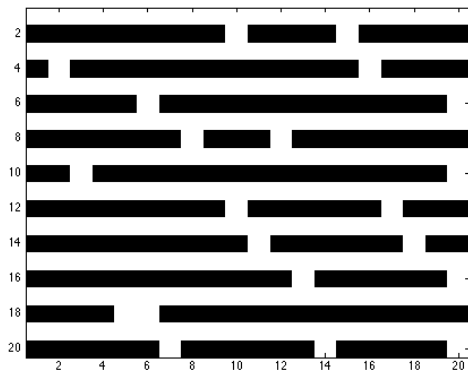
② We update γ_t , given $S(\mathbf{X}_1), \dots, S(\mathbf{X}_N)$

③ We update of the parameters of the policy matrix $P_t = \{p_{t,za}\}$:

$$p_{t,za} = \frac{\sum_{k=1}^N \mathbb{I}_{\{S(\mathbf{x}) \geq \gamma_t\}} \mathbb{I}_{\{\mathbf{x}_k \in \mathcal{X}_{za}\}}}{\sum_{k=1}^N \mathbb{I}_{\{S(\mathbf{x}) \geq \gamma_t\}} \mathbb{I}_{\{\mathbf{x}_k \in \mathcal{X}_z\}}}$$

The maze problem

The maze problem belong to the class of the Markovian decision process.
We are in a 2-dimensional grid world.



The maze problem: the rules

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 .

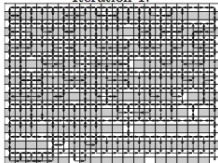
In addition we introduce:

- 1 A small 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 (-50) for the moves in a forbidden direction.

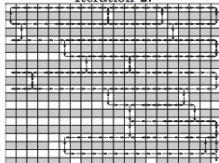
The maze problem: the results

In the paper the authors run the algorithm with the following parameters:
 $\rho = 0.03$, $N = 1000$ and get the solution in few iterations.

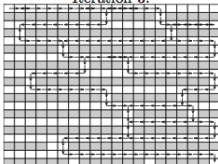
Iteration 1:



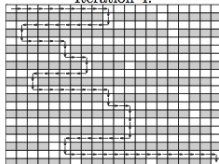
Iteration 2:



Iteration 3:



Iteration 4:



The maze problem: the results

We tried to simulate the previous problem, using the code `maze.m`, but we didn't manage to obtain the same results as in the paper.

The algorithm should work as follows:

- Identifying the *sensible nodes*

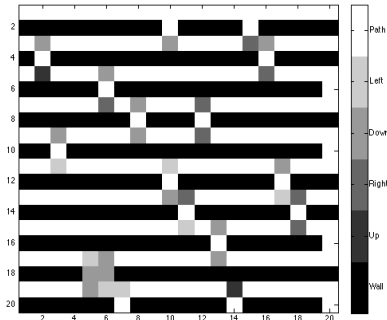


The maze problem: the results

We tried to simulate the previous problem, using the code `maze.m`, but we didn't manage to obtain the same results as in the paper.

The algorithm should work as follows:

- Identifying the *sensible nodes*
- Updating the probabilities and building the optimal path



Other remarks

Alternative performance functions: we recall that in a step of the algorithm we shall estimate $l(\gamma) = \mathbb{E}_{\mathbf{u}}[\mathbb{I}_{\{S(\mathbf{X}) > \gamma\}}]$. We can rewrite it as $l(\gamma) = \mathbb{E}_{\mathbf{u}}[\phi(S(\mathbf{X}), \gamma)]$, where:

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

The paper suggest alternative $\phi(\cdot, \cdot)$, such as $\phi(s, \gamma) = \psi(s)\mathbb{I}_{\{s \geq \gamma\}}$, where $\psi(\cdot)$ is some increasing function.

Numerical evidence suggests $\psi(s) = s$ or $\psi(s) = s^\beta$, but there could be problems with local minima.

Other remarks

Alternative performance functions: we recall that in a step of the algorithm we shall estimate $l(\gamma) = \mathbb{E}_{\mathbf{u}}[\mathbb{I}_{\{S(\mathbf{X}) > \gamma\}}]$. We can rewrite it as $l(\gamma) = \mathbb{E}_{\mathbf{u}}[\phi(S(\mathbf{X}), \gamma)]$, where:

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

The paper suggest alternative $\phi(\cdot, \cdot)$, such as $\phi(s, \gamma) = \psi(s)\mathbb{I}_{\{s \geq \gamma\}}$, where $\psi(\cdot)$ is some increasing function.

Numerical evidence suggests $\psi(s) = s$ or $\psi(s) = s^\beta$, but there could be problems with local minima.

Fully Adaptive CE algorithm: Modification of the standard algorithm, where the sample size and the parameter ρ are updated adaptively at each iteration t of the algorithm, i.e., $N = N_t$ and $\rho = \rho_t$.

Conclusions

The Cross-entropy method appears to be:

- *A simple, versatile and easy usable tool.*
- Robust with respect to his parameters (N, ρ). I tried to simulate the first example with other values of ρ and the results are the same up to the 7th decimal digit ($1.37 \cdot 10^{-5}$ instead of $1.33 \cdot 10^{-5}$).
- With a wide range of parameters N, ρ, \mathbf{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.*
- Problem-independent because it is based on some well known classical probabilistic and statistical principles.