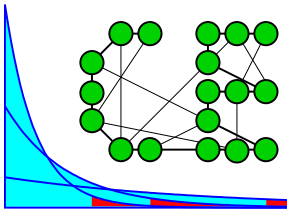
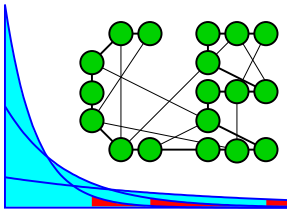


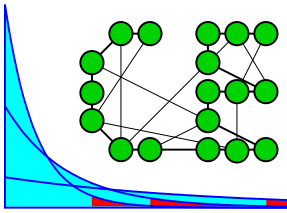
The Cross-Entropy Method for Mathematical Programming

*Dirk P. Kroese Reuven Y. Rubinstein

*Department of Mathematics, The University of Queensland, Australia
Faculty of Industrial Engineering and Management, Technion, Israel

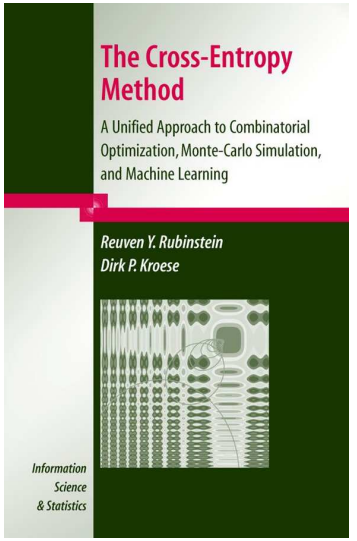


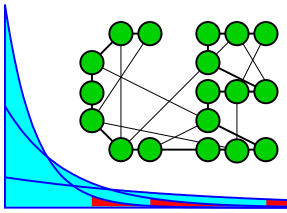
Contents



CE Matters

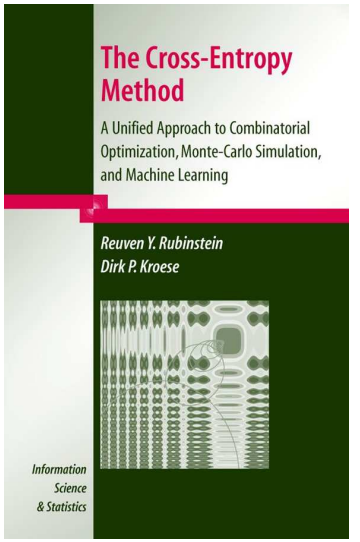
Book: R.Y. Rubinstein and D.P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte Carlo Simulation and Machine Learning*, Springer-Verlag, New York, 2004.



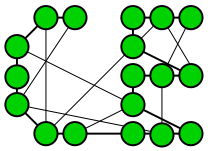


CE Matters

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

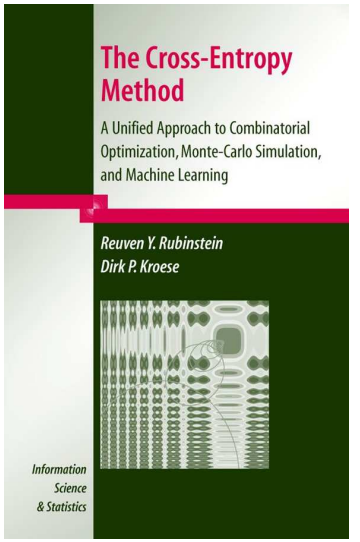


Special Issue: *Annals of Operations Research* (Jan 2005).



CE Matters

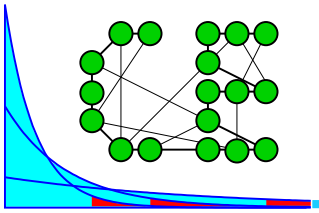
Book: R.Y. Rubinstein and D.P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte Carlo Simulation and Machine Learning*, Springer-Verlag, New York, 2004.



Special Issue: *Annals of Operations Research* (Jan 2005).

The CE **home page**:

`http://www.cemethod.org`



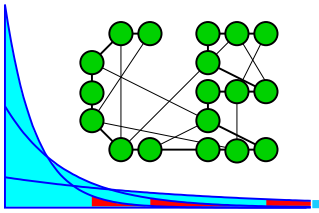
Introduction

The Cross-Entropy Method was originally developed as a simulation method for the estimation of *rare event* probabilities:

$$\text{Estimate} \quad \mathbb{P}(S(\boldsymbol{X}) \geq \gamma)$$

$\boldsymbol{X}$: random vector/process taking values in some set $\mathcal{X}$.

S: real-valued function on $\mathcal{X}$.



Introduction

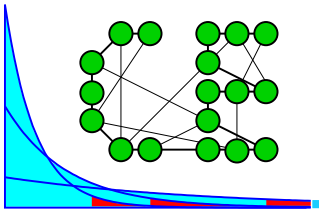
The Cross-Entropy Method was originally developed as a simulation method for the estimation of *rare event* probabilities:

Estimate $\mathbb{P}(S(\boldsymbol{X}) \geq \gamma)$

$\boldsymbol{X}$: random vector/process taking values in some set $\mathcal{X}$.
S: real-valued function on $\mathcal{X}$.

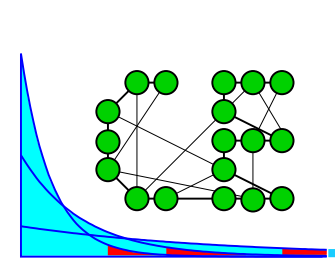
It was soon realised that the CE Method could also be used as an *optimization* method:

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

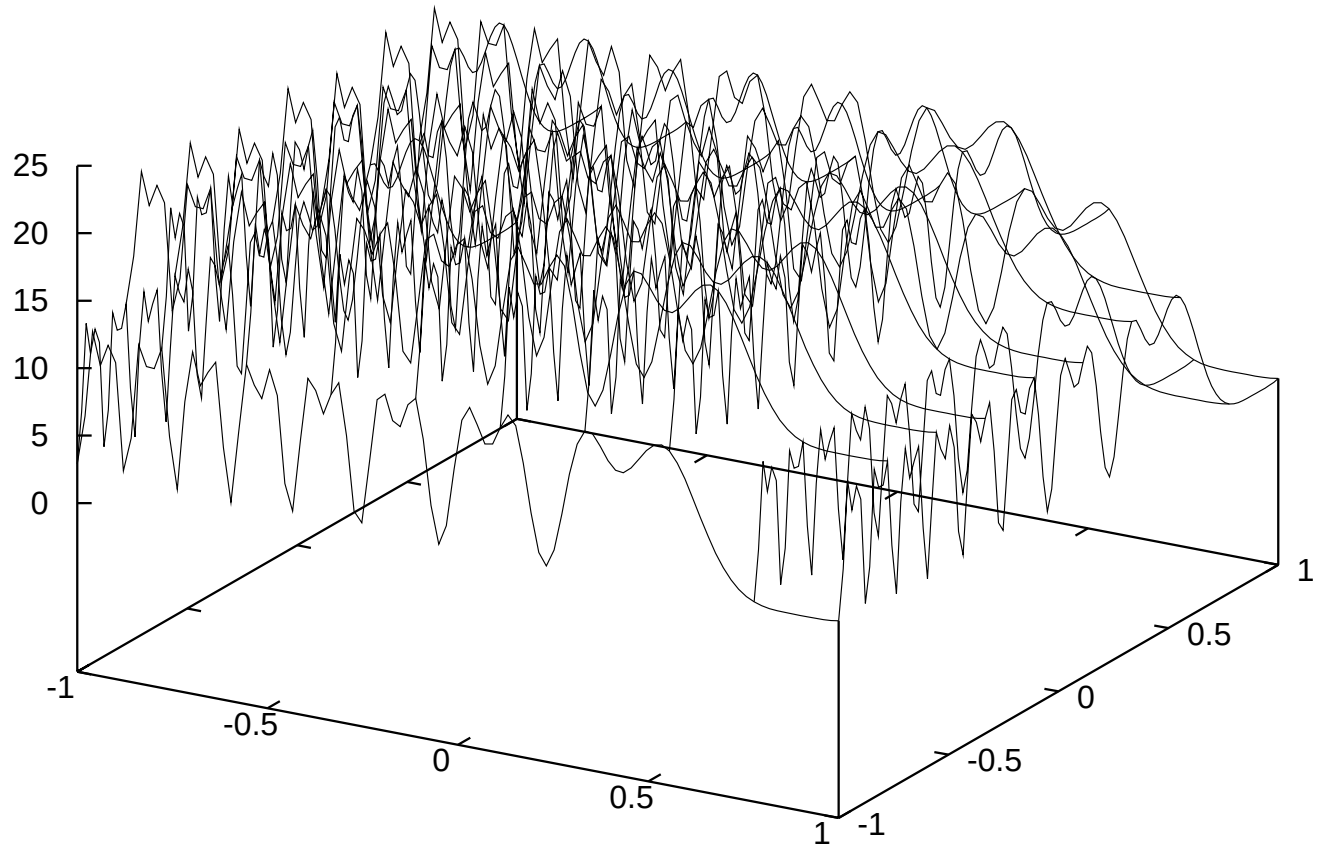


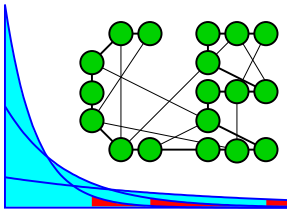
Some Applications

- Combinatorial Optimization (e.g., Travelling Salesman, Maximal Cut and Quadratic Assignment Problems)
- Noisy Optimization (e.g., Buffer Allocation, Financial Engineering)
- Multi-Extremal Continuous Optimization
- Pattern Recognition, Clustering and Image Analysis
- Production Lines and Project Management
- Network Reliability Estimation
- Vehicle Routing and Scheduling
- DNA Sequence Alignment



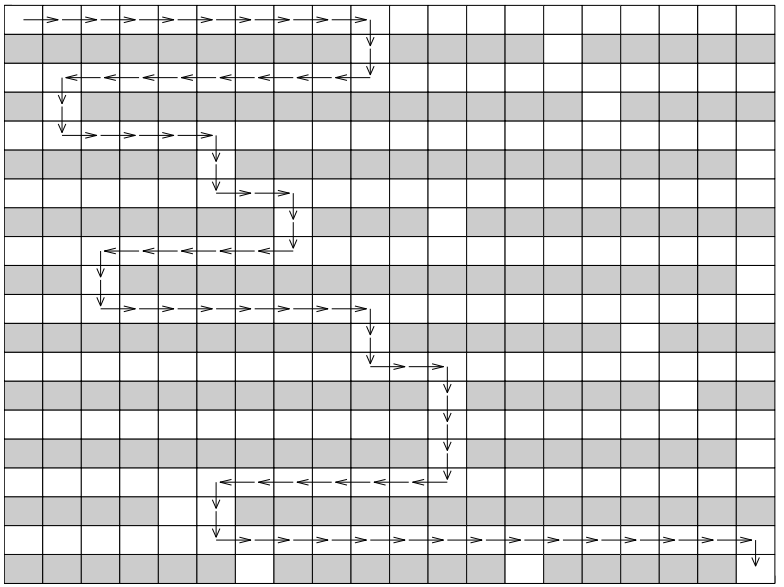
A Multi-extremal Function

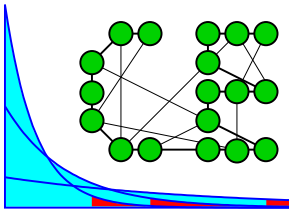




A Maze Problem

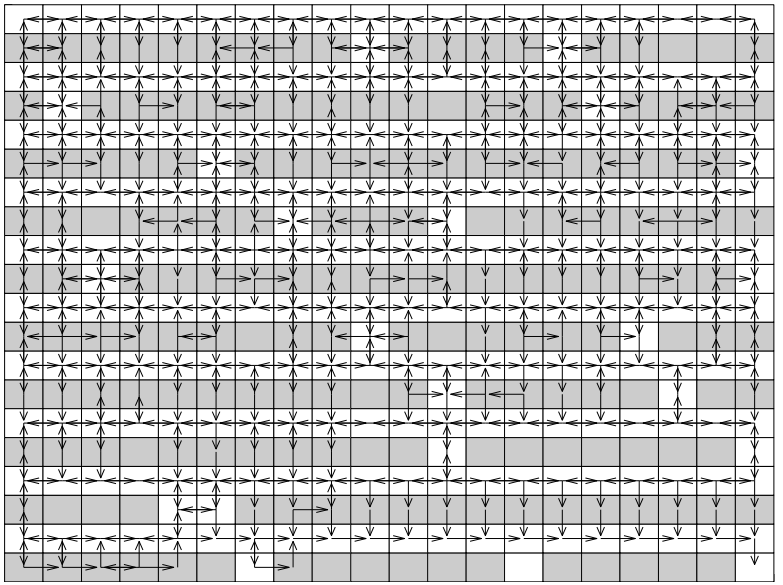
The Optimal Trajectory

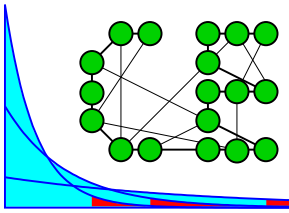




A Maze Problem

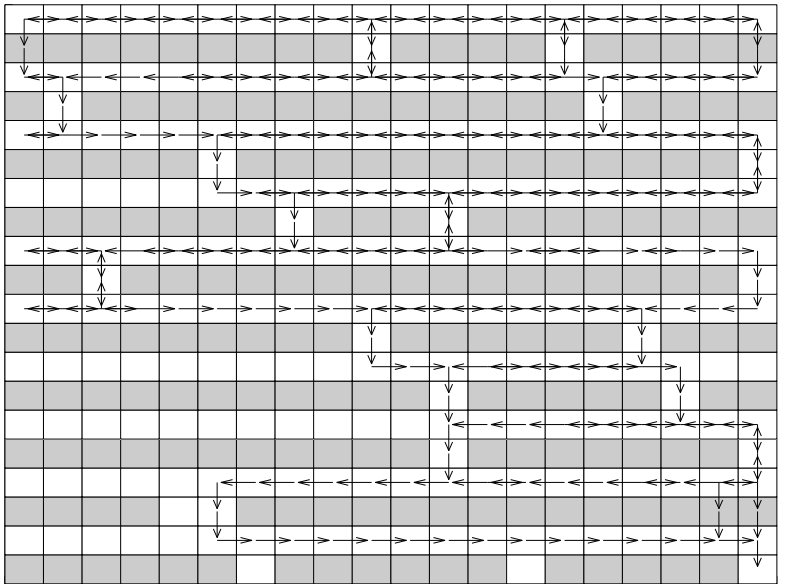
Iteration 1:

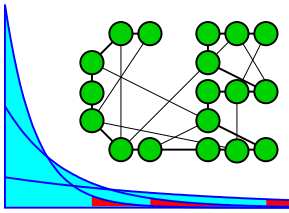




A Maze Problem

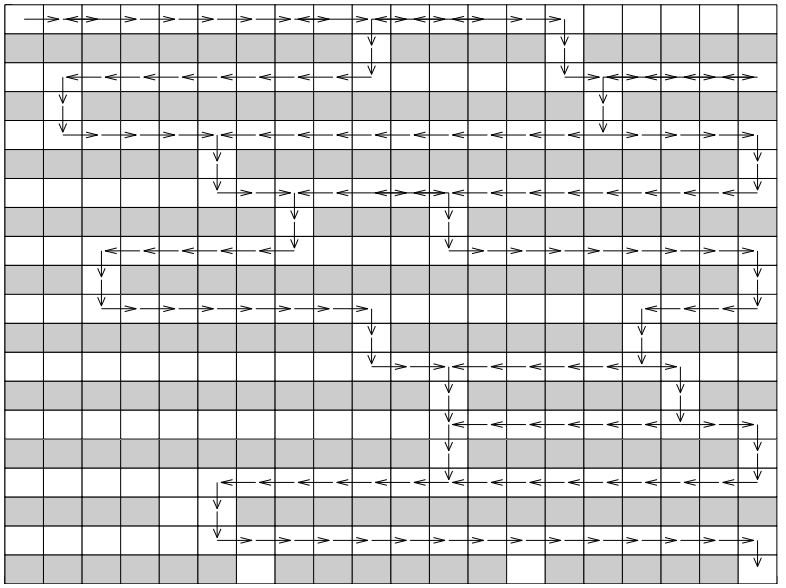
Iteration 2:

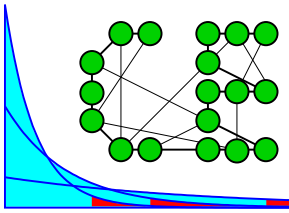




A Maze Problem

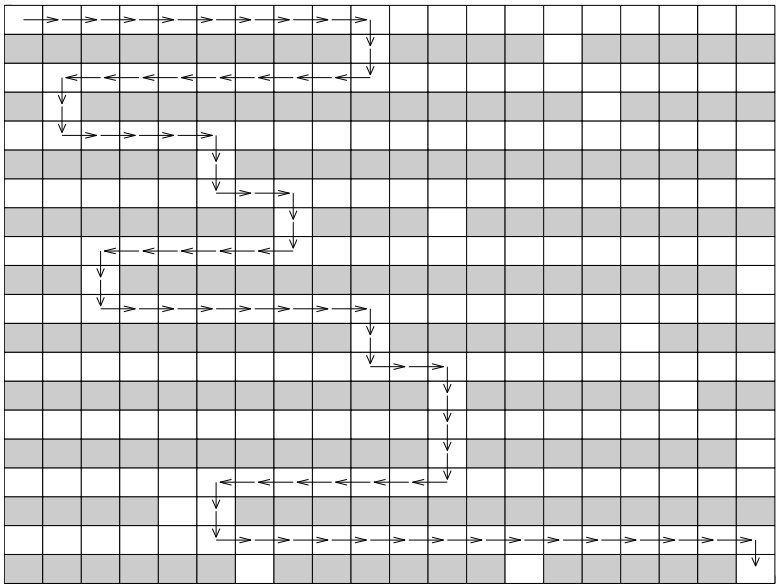
Iteration 3:

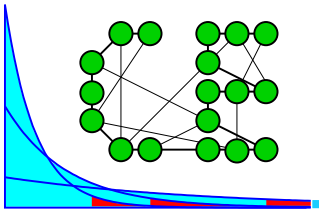




A Maze Problem

Iteration 4:





The Optimisation Setting

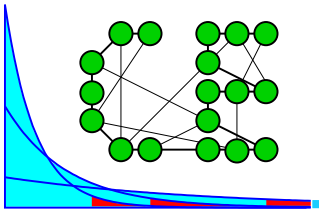
Many real-world optimisation problems can be formulated mathematically as either (or both)

1. Locate some element $\boldsymbol{x}^*$ in a set $\mathcal{X}$ such that

$$S(\boldsymbol{x}^*) \geq S(\boldsymbol{x}) \text{ for all } \boldsymbol{x} \in \mathcal{X},$$

   where S is a objective function or performance measure defined on $\mathcal{X}$.

2. Find $\gamma^* = S(\boldsymbol{x}^*)$, the globally maximal value of the function.



The Optimisation Setting

Many real-world optimisation problems can be formulated mathematically as either (or both)

1. Locate some element $\boldsymbol{x}^*$ in a set $\mathcal{X}$ such that

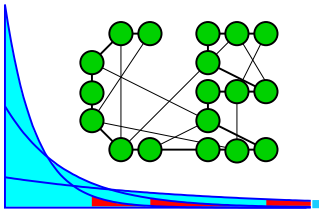
$$S(\boldsymbol{x}^*) \geq S(\boldsymbol{x}) \text{ for all } \boldsymbol{x} \in \mathcal{X},$$

   where S is a objective function or performance measure defined on $\mathcal{X}$.

2. Find $\gamma^* = S(\boldsymbol{x}^*)$, the globally maximal value of the function.

$\mathcal{X}$ discrete: combinatorial optimisation problem

$\mathcal{X}$ continuous: continuous optimisation problem.

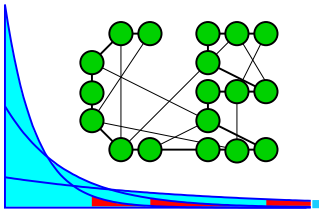


A New Approach: CE

Instead of locating optimal solutions to a particular problem directly, the CE method aims to

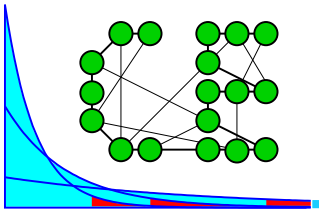
locate an optimal sampling distribution

The sampling distribution is optimal if only optimal solutions can be sampled from it.



General CE Procedure

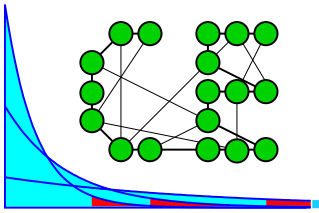
First, provide the state space $\mathcal{X}$ (whether the states are coordinates, permutations, or some other mathematical entities) over which the problem is defined, along with a performance measure S on this space.



General CE Procedure

First, provide the state space $\mathcal{X}$ (whether the states are coordinates, permutations, or some other mathematical entities) over which the problem is defined, along with a performance measure S on this space.

Second, formulate the parameterized sampling distribution to generate objects $\boldsymbol{X} \in \mathcal{X}$. Then, iterate the following steps:

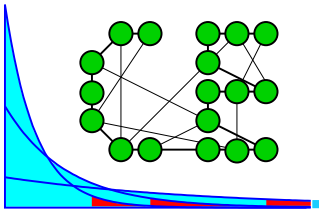


General CE Procedure

First, provide the state space $\mathcal{X}$ (whether the states are coordinates, permutations, or some other mathematical entities) over which the problem is defined, along with a performance measure S on this space.

Second, formulate the parameterized sampling distribution to generate objects $\boldsymbol{X} \in \mathcal{X}$. Then, iterate the following steps:

- *Generate a random sample* of states $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N \in \mathcal{X}$.

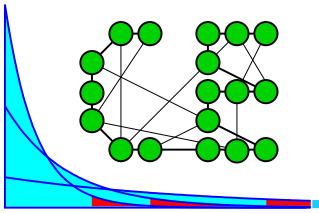


General CE Procedure

First, provide the state space $\mathcal{X}$ (whether the states are coordinates, permutations, or some other mathematical entities) over which the problem is defined, along with a performance measure S on this space.

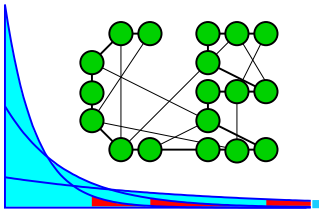
Second, formulate the parameterized sampling distribution to generate objects $\boldsymbol{X} \in \mathcal{X}$. Then, iterate the following steps:

- *Generate a random sample* of states $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N \in \mathcal{X}$.
- *Update the parameters* of the random mechanism (obtained via CE minimization), in order to produce a better sample in the next iteration.



General CE Algorithm

1. Initialise the parameters of the algorithm.
2. Generate a random sample from the sampling distribution.
3. Update the parameters of the sampling distribution on the basis of the best scoring samples, the so-called elite samples.
   This step involves the Cross-Entropy distance.
4. Repeat from Step 2 until some stopping criterion is met.

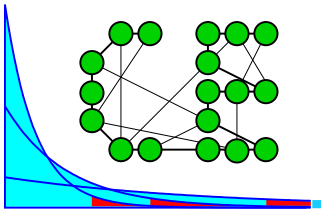


Degenerate Sampling Distribution

Suppose that there is a unique solution $\boldsymbol{x}^*$ to the problem.

Then, the degenerate distribution assigns all of the probability mass (in the discrete case) or density (in the continuous case) to this point in $\mathcal{X}$.

It is exactly the degenerate distribution from which we wish to ultimately sample, in order to obtain an optimal solution.

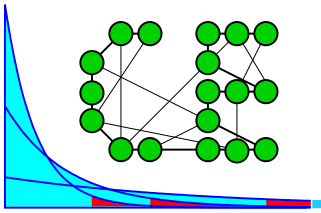


Detailed Generic CE Algorithm

1. Choose an initial parameter vector, $\boldsymbol{v}_0$. Set $t = 1$.
2. Generate $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_N$ from the density $f(\cdot; \boldsymbol{v}_{t-1})$
3. Compute the sample $(1-\rho)$-quantile, $\hat{\gamma}_t = S_{(\lceil (1-\rho)N \rceil)}$, locate the elite samples $\mathcal{E} = \{\boldsymbol{X}_i \ : \ S(\boldsymbol{X}_i) \geq \hat{\gamma}_t\}$, and determine

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

   Set the current estimate for the optimal parameter vector, $\boldsymbol{v}_t$ to: $\boldsymbol{v}_t = \alpha \tilde{\boldsymbol{v}}_t + (1-\alpha)\boldsymbol{v}_{t-1}$.
4. If the stopping criterion is met, stop; otherwise set $t = t + 1$, and return to Step 2.

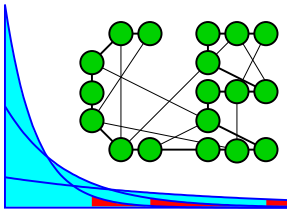


Maximum Likelihood

There is a connection between the Cross-Entropy method and Maximum Likelihood Estimation, which implies a very simple way of updating the parameter vector:

> take the maximum likelihood estimate of the parameter vector based on the elite samples.

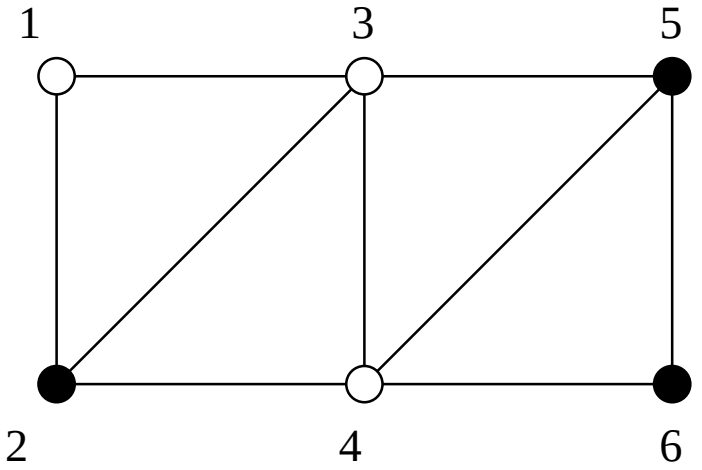
Thus, the sampling distribution comes “closer” at each iteration to the degenerate distribution, which is the optimal sampling distribution for the problem.

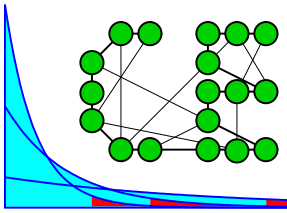


Example: The Max-Cut Problem

Consider a weighted graph G with node set $V = \{1, \ldots, n\}$. 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 maximised.

Example:



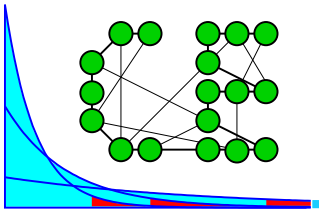


Cost matrix (assume symmetric):

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

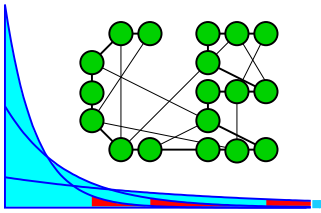
$\{V_1, V_2\} = \{\{1,3,4\},\{2,5,6\}\}$ is a possible cut. The cost of the cut is

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



Random Cut Vector

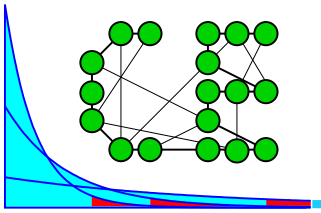
We can represent a cut via a cut vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, where $x_i = 1$ if node i belongs to same partition as 1, and 0 else.



Random Cut Vector

We can represent a cut via a cut vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, where $x_i = 1$ if node i belongs to same partition as 1, and 0 else.

For example, the cut $\{\{1, 3, 4\}, \{2, 5, 6\}\}$ can be represented via the cut vector $(1, 0, 1, 1, 0, 0)$.

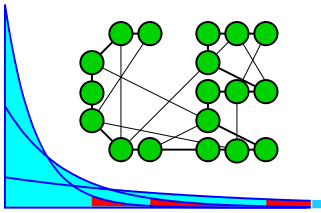


Random Cut Vector

We can represent a cut via a cut vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, where $x_i = 1$ if node i belongs to same partition as 1, and 0 else.

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 $\boldsymbol{x} = (1, x_2, \ldots, x_n)$.



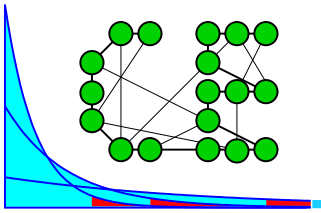
Random Cut Vector

We can represent a cut via a cut vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, where $x_i = 1$ if node i belongs to same partition as 1, and 0 else.

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 $\boldsymbol{x} = (1, x_2, \ldots, x_n)$.

Let $S(\boldsymbol{x})$ be the corresponding cost of the cut.



Random Cut Vector

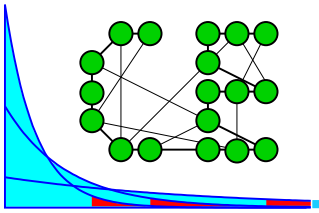
We can represent a cut via a cut vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, where $x_i = 1$ if node i belongs to same partition as 1, and 0 else.

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 $\boldsymbol{x} = (1, x_2, \ldots, x_n)$.

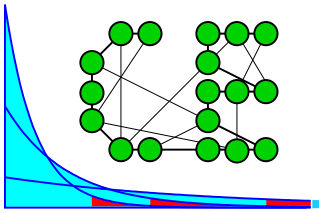
Let $S(\boldsymbol{x})$ be the corresponding cost of the cut.

We wish to maximise $S(\boldsymbol{x})$ via the CE method.



Generation and Updating Formulas

Generation of cut vectors: The most natural and easiest way to generate the cut vectors $\boldsymbol{X}_i = (1, X_{i1}, \ldots, X_{in})$ is to let $X_{i2}, \ldots, X_{in}$ be independent Bernoulli random variables with success probabilities $p_2, \ldots, p_n$.

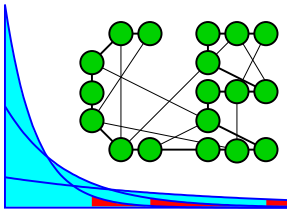


Generation and Updating Formulas

Generation of cut vectors: The most natural and easiest way to generate the cut vectors $\boldsymbol{X}_i = (1, X_{i1}, \ldots, X_{in})$ is to let $X_{i2}, \ldots, X_{in}$ be independent Bernoulli random variables with success probabilities $p_2, \ldots, p_n$.

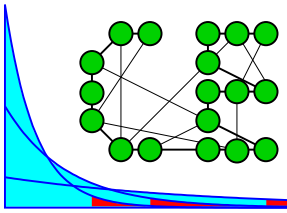
Updating formulas: From CE minimization: the updated probabilities are the maximum likelihood estimates of the $|\mathcal{E}| = \rho N$ elite samples:

$$\hat{p}_{t,j} = \frac{\sum_{\boldsymbol{X}_i \in \mathcal{E}} X_{ij}}{|\mathcal{E}|}, \quad j = 2, \ldots, n \; .$$



Algorithm

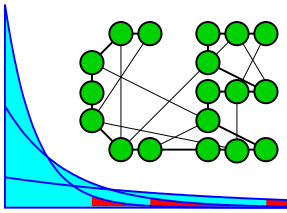
1 Start with $\hat{\boldsymbol{p}}_0 = (1, 1/2, \ldots, 1/2)$. Let $t := 1$.



Algorithm

1 **Start with $\hat{\boldsymbol{p}}_0 = (1, 1/2, \ldots, 1/2)$. Let $t := 1$.**

2 **Generate: Draw $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ from $\mathsf{Ber}(\hat{\boldsymbol{p}}_t)$. Let $\hat{\gamma}_t$ be the worst performance of the elite performances.**



Algorithm

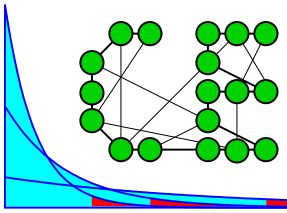
1 **Start with $\hat{\boldsymbol{p}}_0 = (1, 1/2, \ldots, 1/2)$. Let $t := 1$.**

2 **Generate: Draw $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ from $\mathsf{Ber}(\hat{\boldsymbol{p}}_t)$. Let $\hat{\gamma}_t$ be the worst performance of the elite performances.**

3 **Update $\hat{\boldsymbol{p}}_t$: Use the same elite sample to calculate**

$$\hat{p}_{t,j} = \frac{\sum_{\boldsymbol{X}_i \in \mathcal{E}} X_{ij}}{|\mathcal{E}|}, \quad j = 2, \ldots, n \ ,$$

where $\boldsymbol{X}_i = (1, X_{i2}, \ldots, X_{in})$, and increase t by 1.



Algorithm

1 **Start with $\hat{\boldsymbol{p}}_0 = (1, 1/2, \ldots, 1/2)$. Let $t := 1$.**

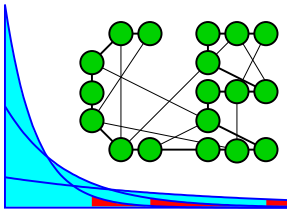
2 **Generate: Draw $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ from $\mathsf{Ber}(\hat{\boldsymbol{p}}_t)$. Let $\hat{\gamma}_t$ be the worst performance of the elite performances.**

3 **Update $\hat{\boldsymbol{p}}_t$: Use the same elite sample to calculate**

$$\hat{p}_{t,j} = \frac{\sum_{\boldsymbol{X}_i \in \mathcal{E}} X_{ij}}{|\mathcal{E}|}, \quad j = 2, \ldots, n \ ,$$

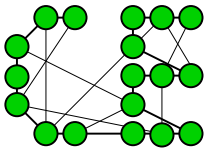
where $\boldsymbol{X}_i = (1, X_{i2}, \ldots, X_{in})$, and increase t by 1.

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



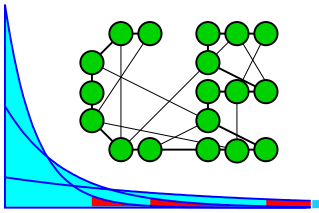
Example

- Results for the case with $n = 400, m = 200$ nodes are given next.



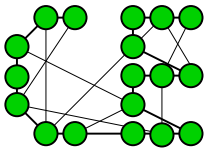
Example

- Results for the case with $n = 400, m = 200$ nodes are given next.
- Parameters: $\rho = 0.1$, $N = 1000$ (thus $|\mathcal{E}| = 100$).



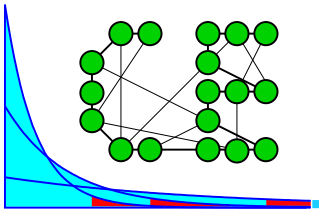
Example

- Results for the case with $n = 400, m = 200$ nodes are given next.
- Parameters: $\rho = 0.1$, $N = 1000$ (thus $|\mathcal{E}| = 100$).
- The CPU time was only 100 seconds (Matlab, pentium III, 500 Mhz).

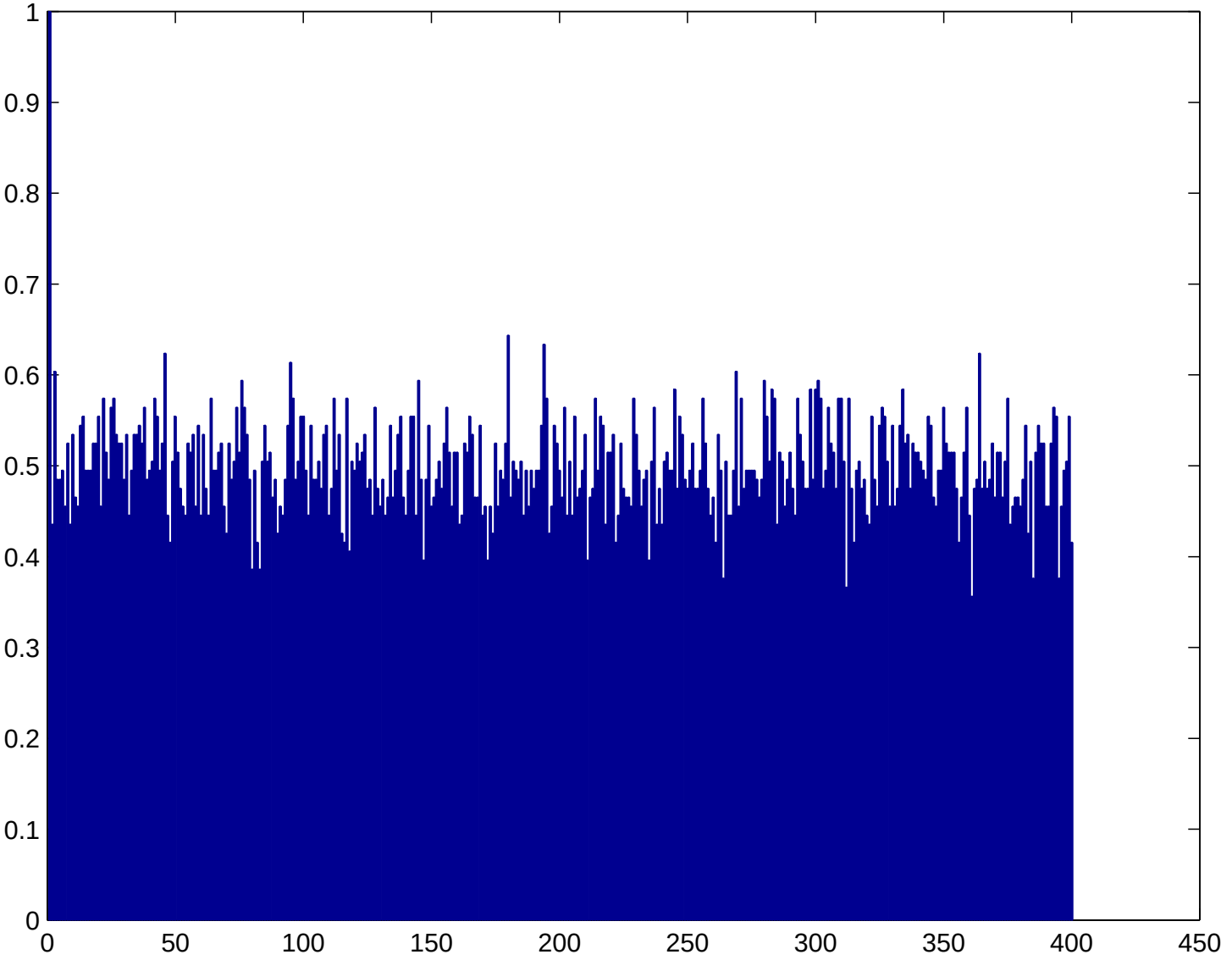


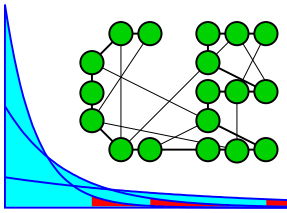
Example

- Results for the case with $n = 400, m = 200$ nodes are given next.
- Parameters: $\rho = 0.1$, $N = 1000$ (thus $|\mathcal{E}| = 100$).
- The CPU time was only 100 seconds (Matlab, pentium III, 500 Mhz).
- The CE algorithm converges quickly, yielding the exact optimal solution in 22 iterations.

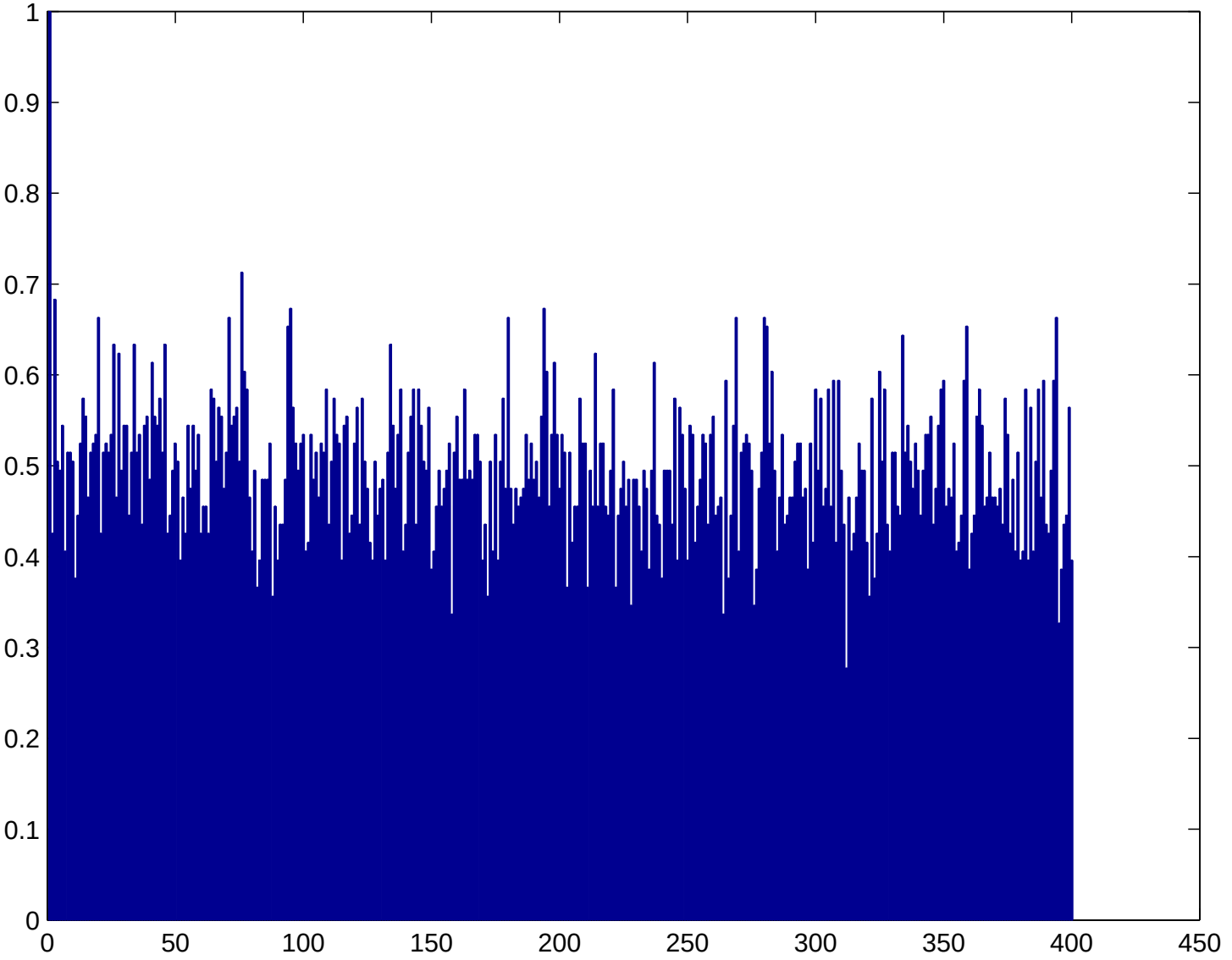


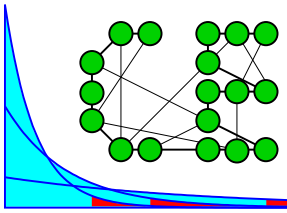
Max-Cut



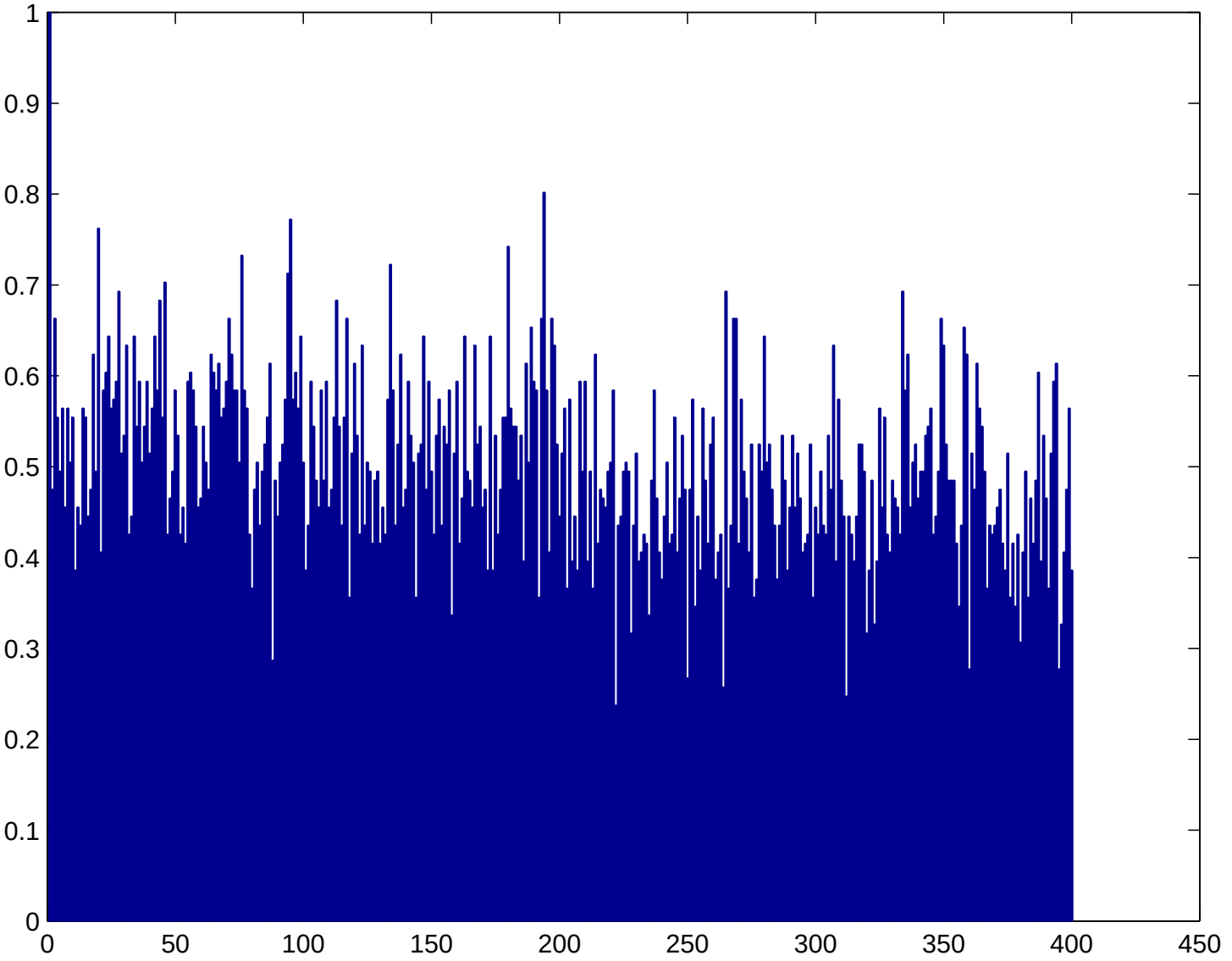


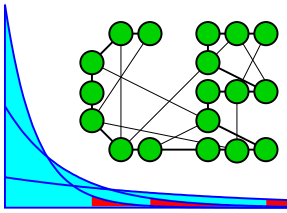
Max-Cut



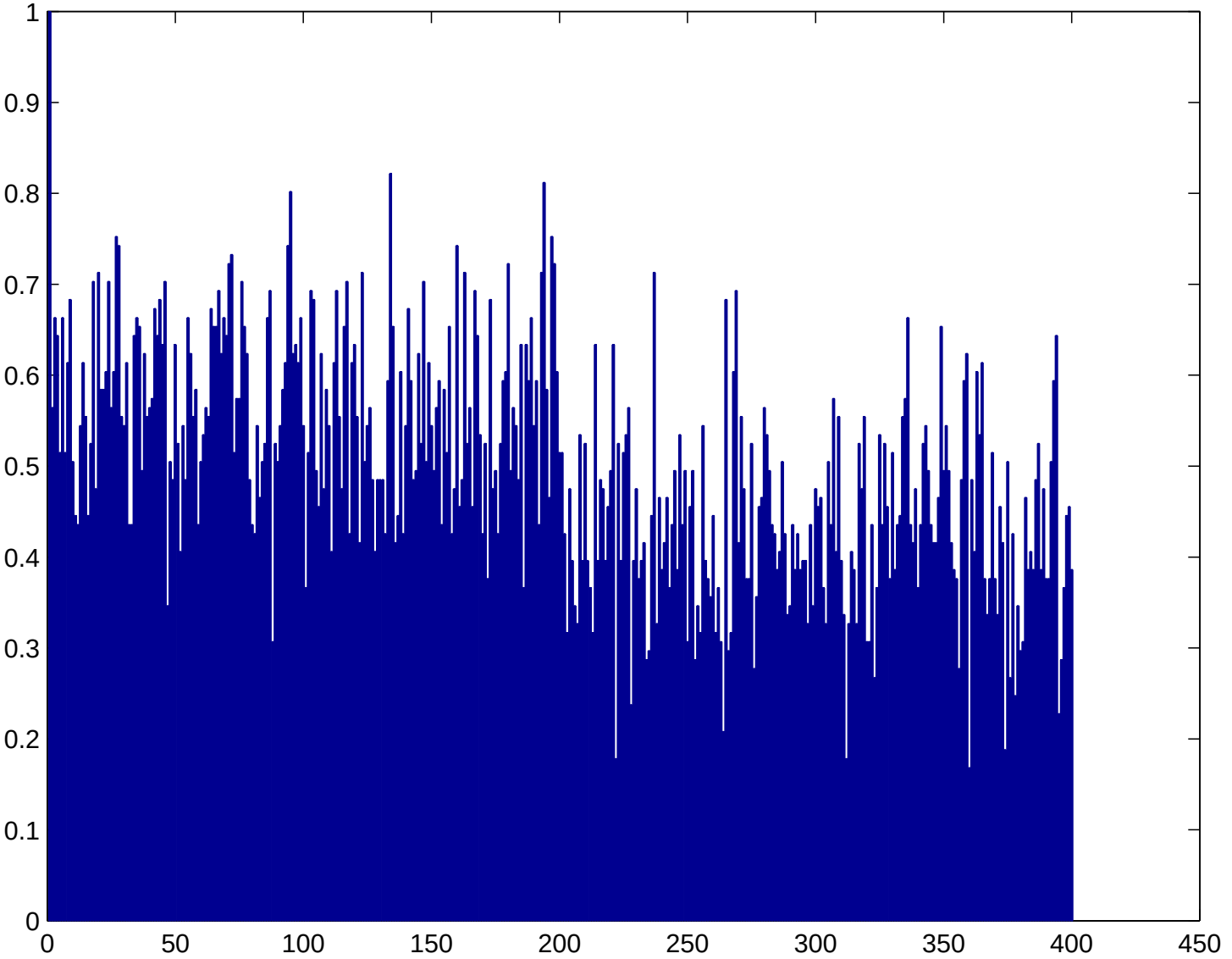


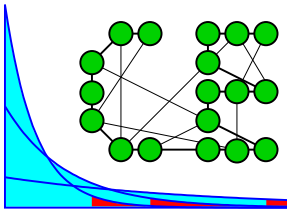
Max-Cut



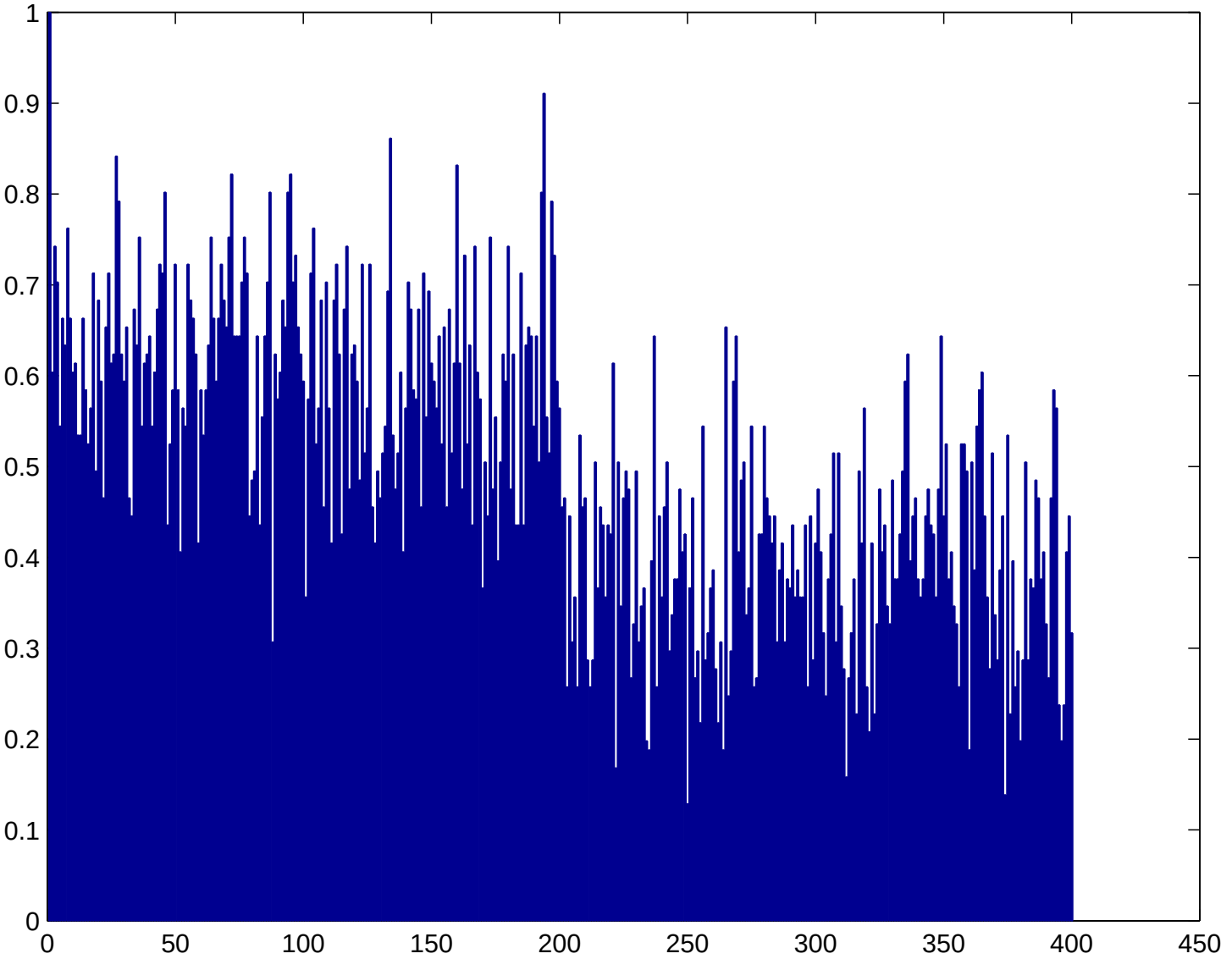


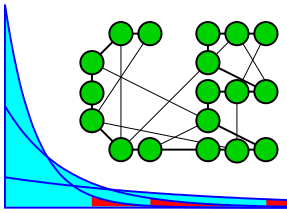
Max-Cut



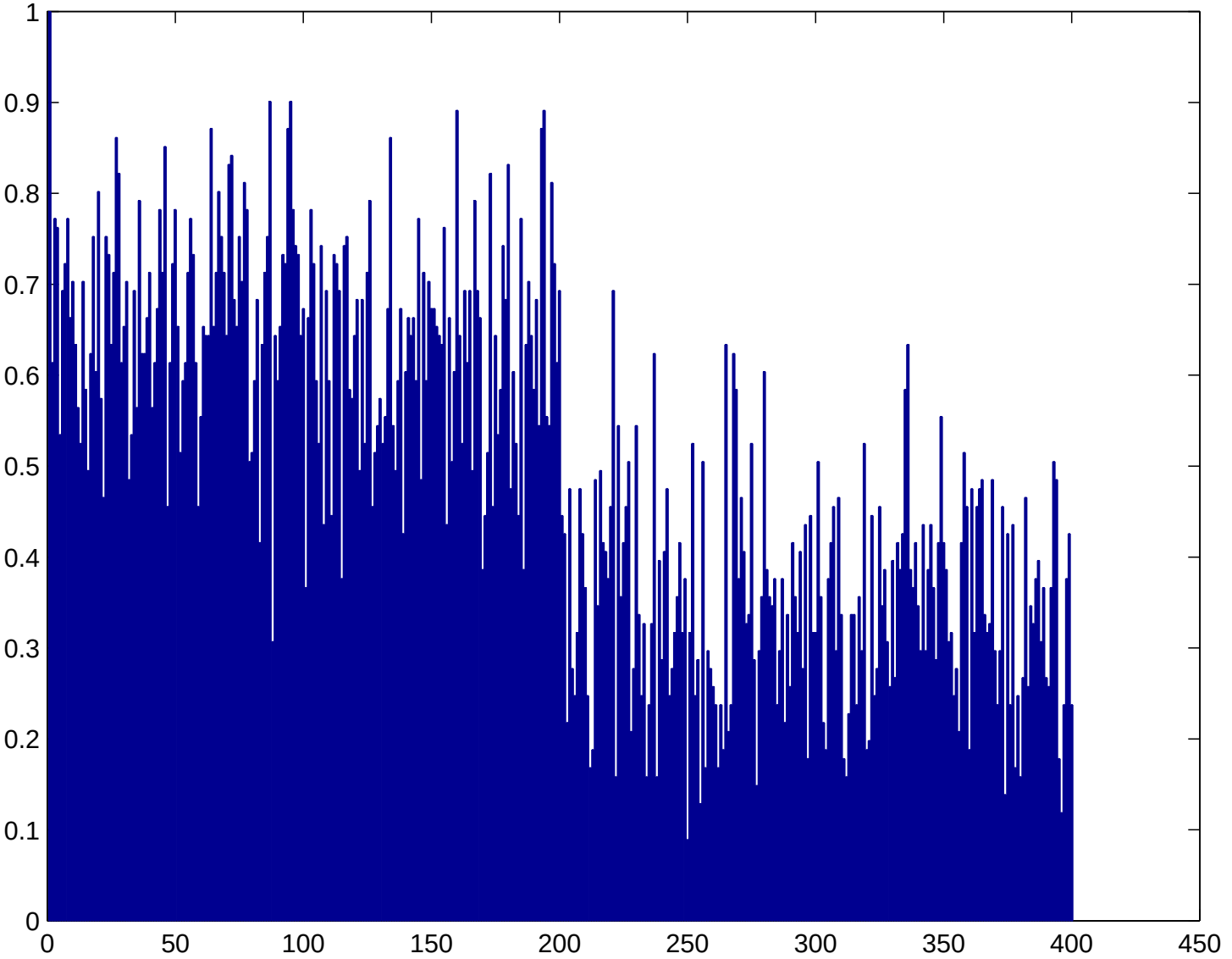


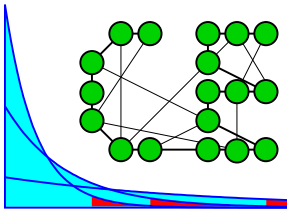
Max-Cut



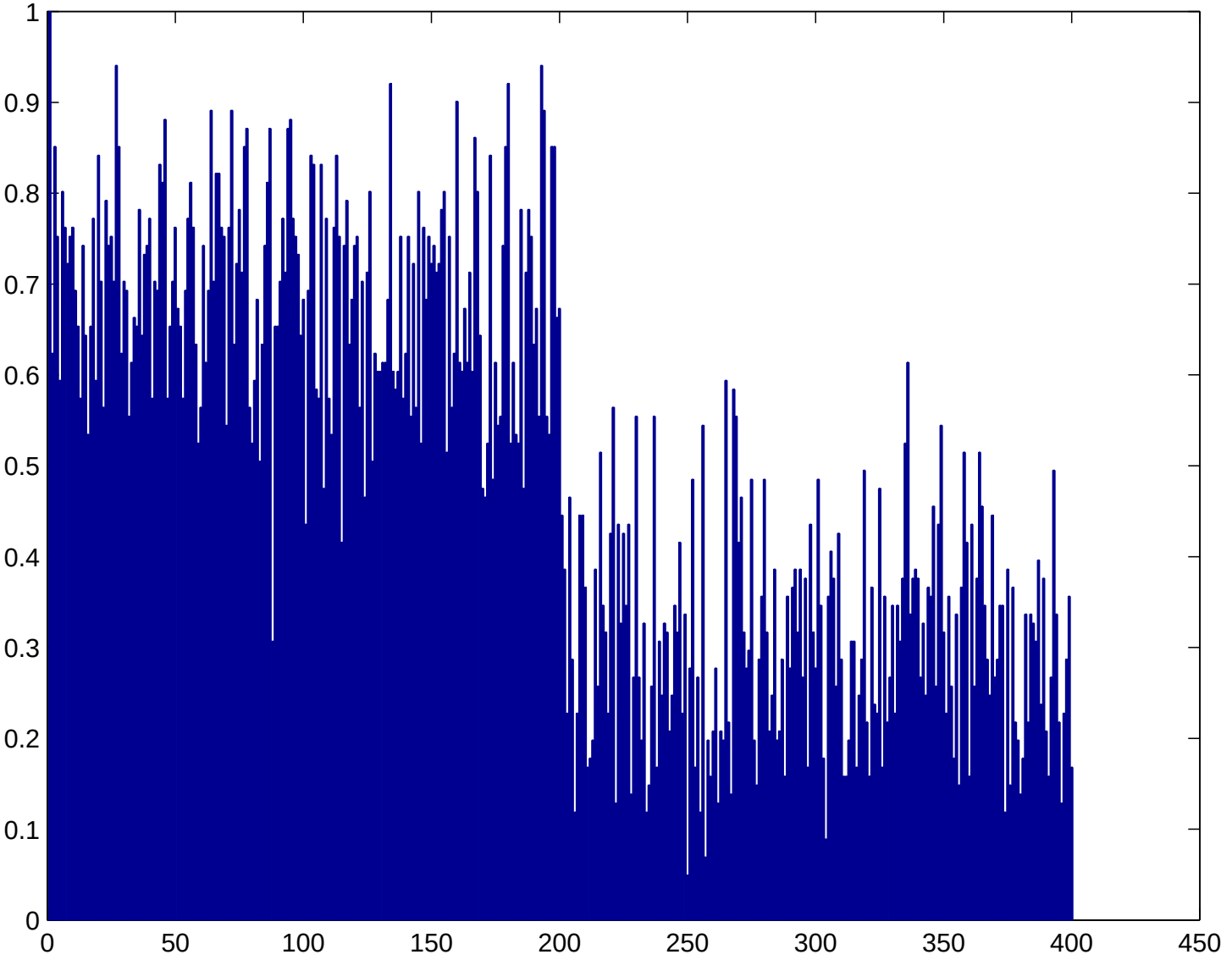


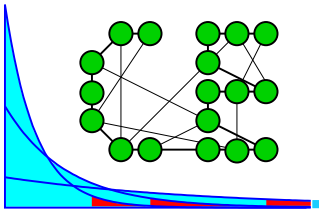
Max-Cut



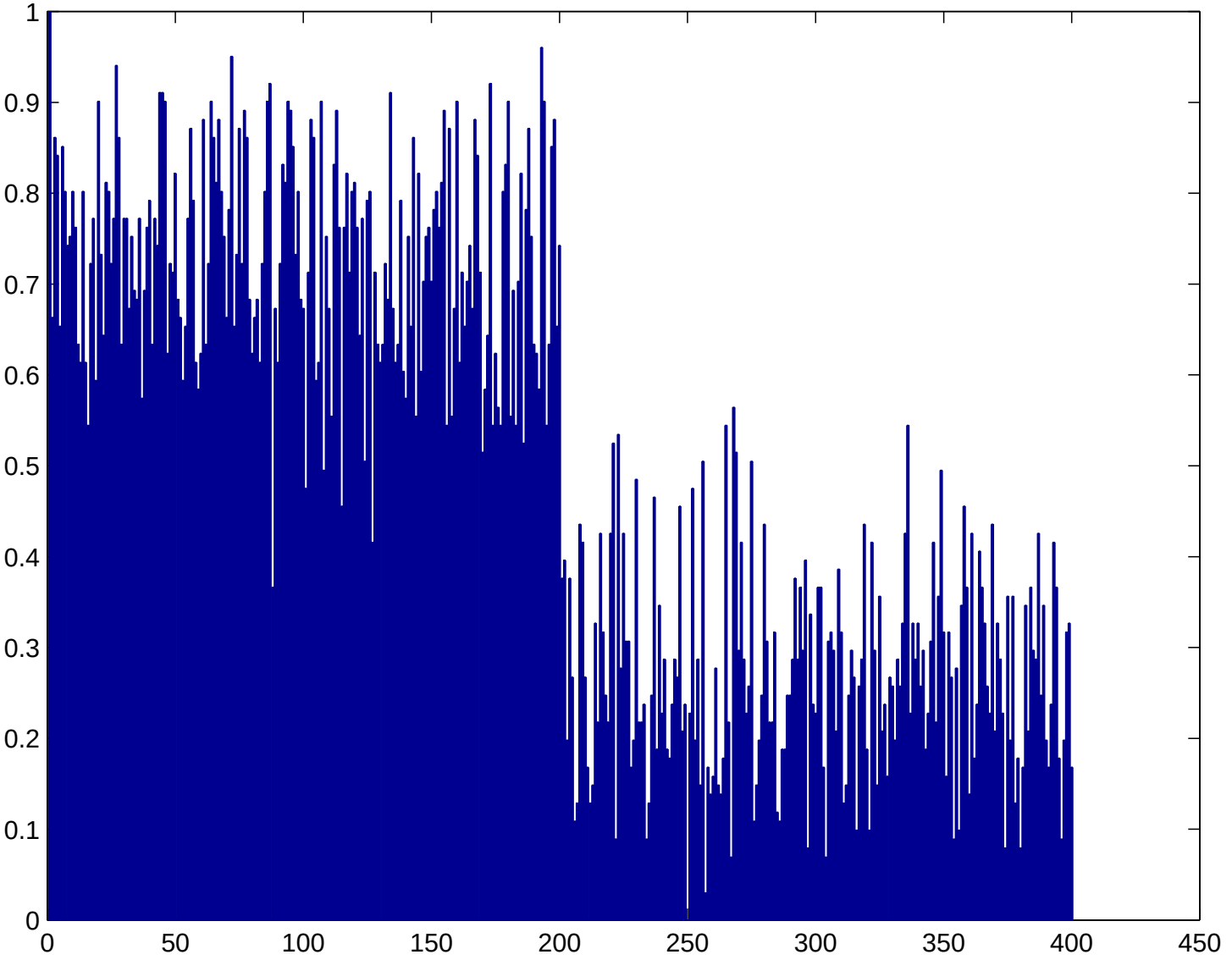


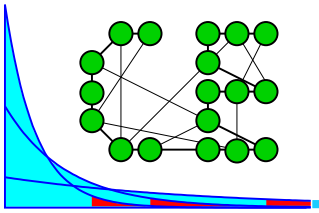
Max-Cut



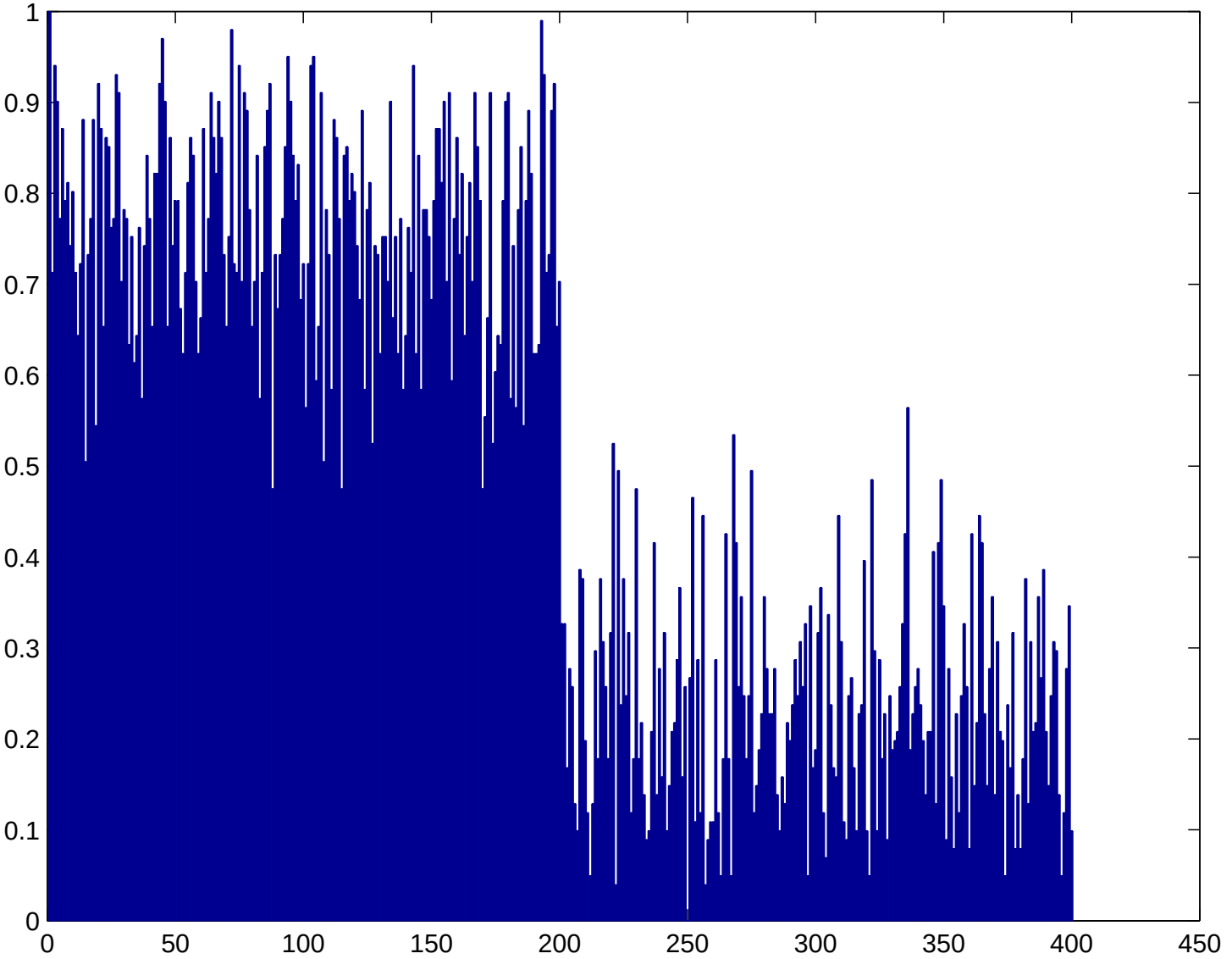


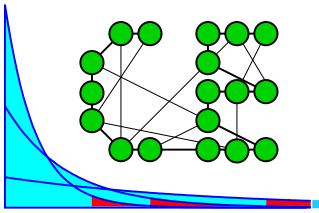
Max-Cut



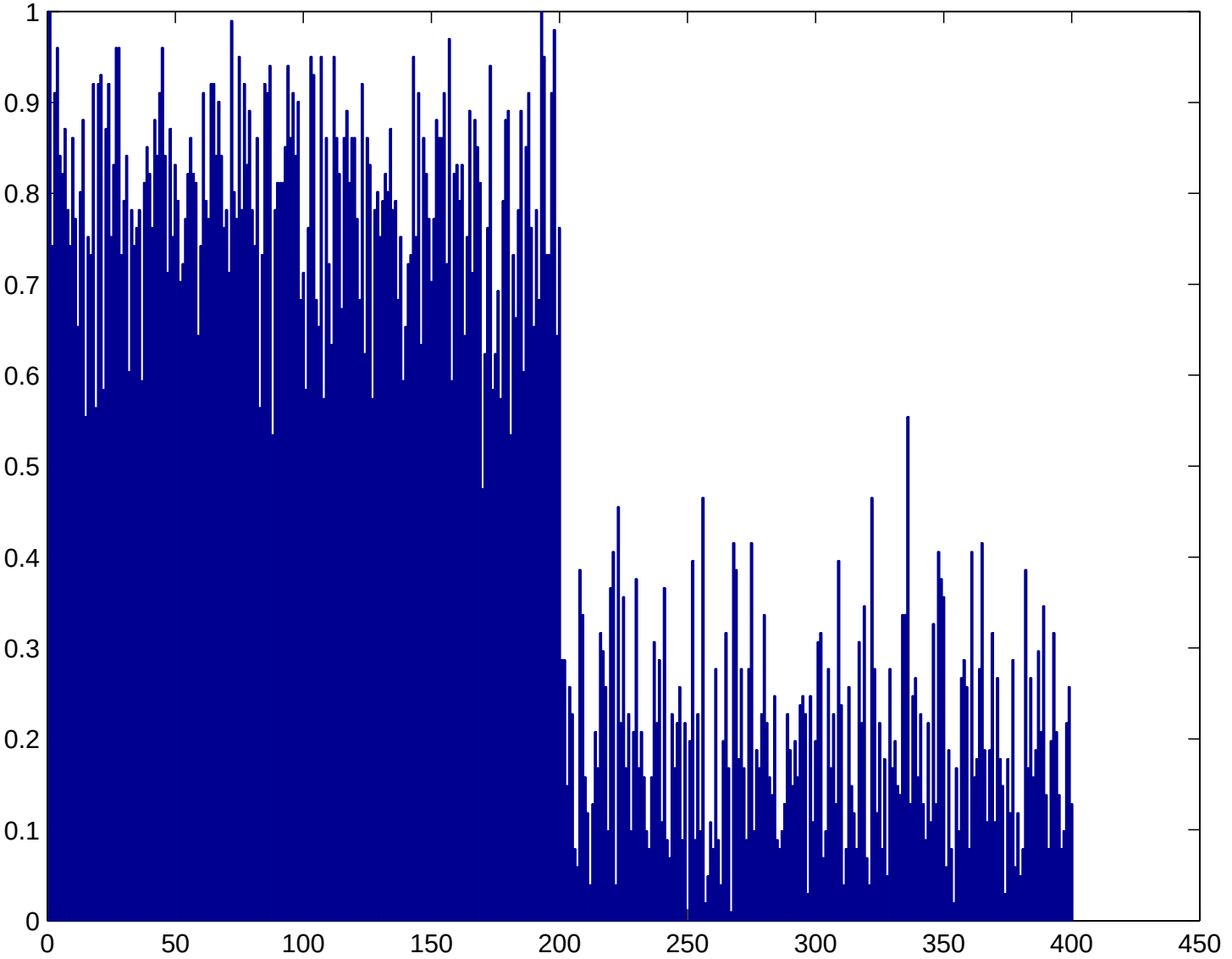


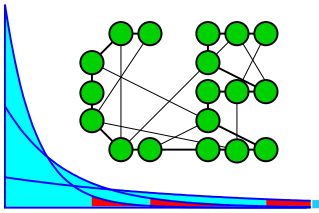
Max-Cut



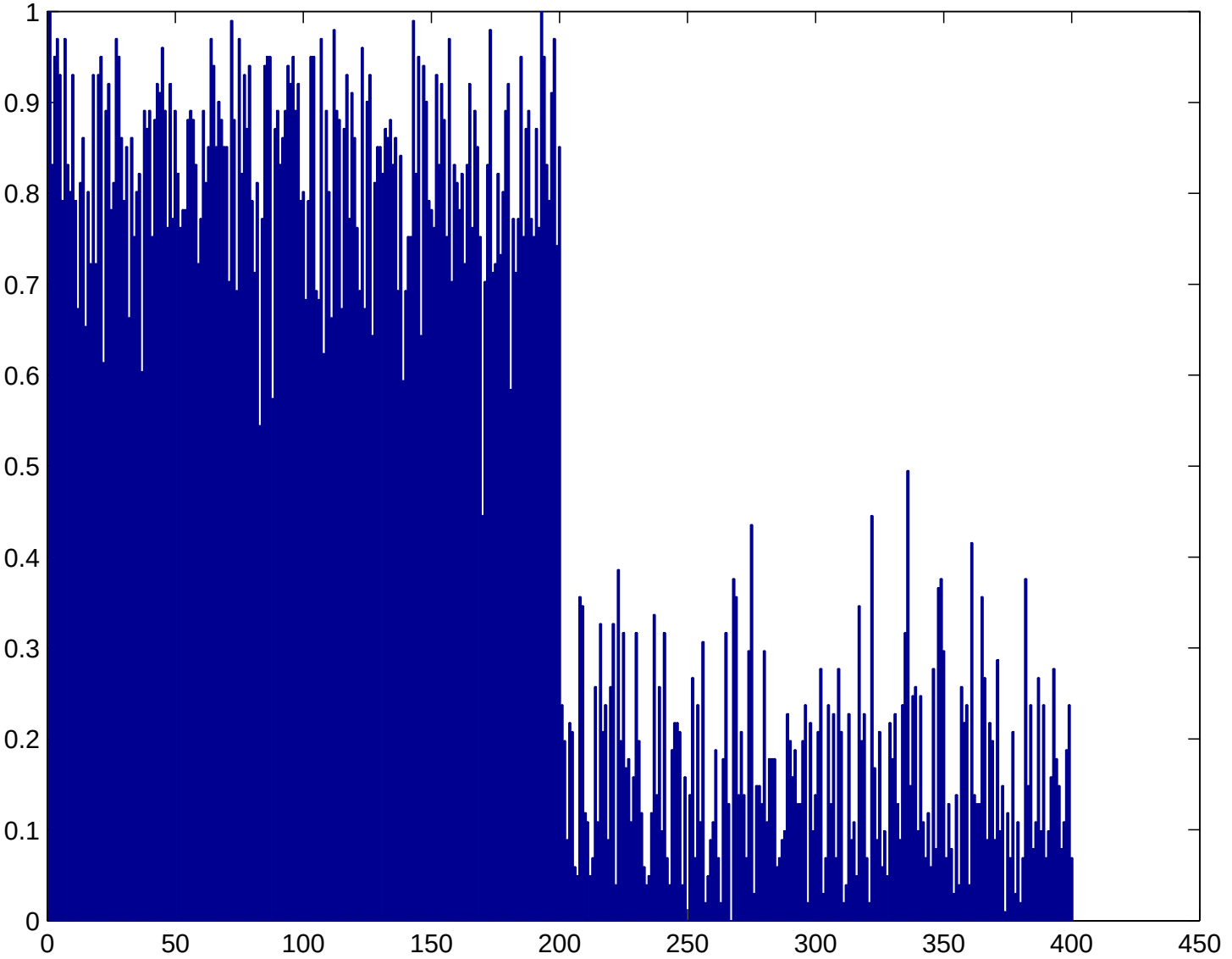


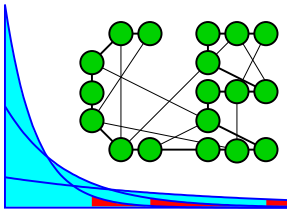
Max-Cut



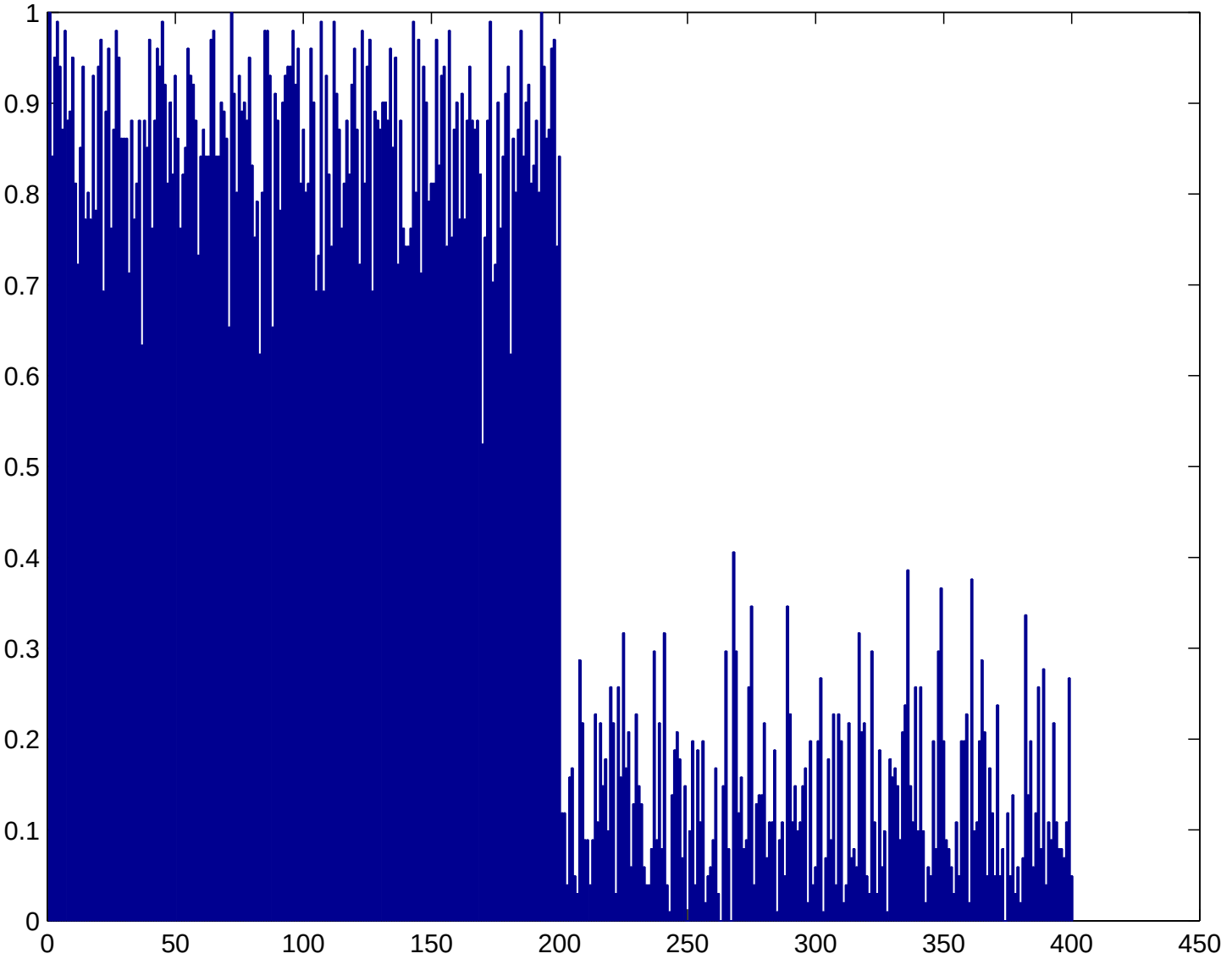


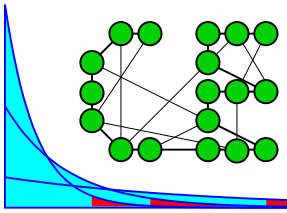
Max-Cut



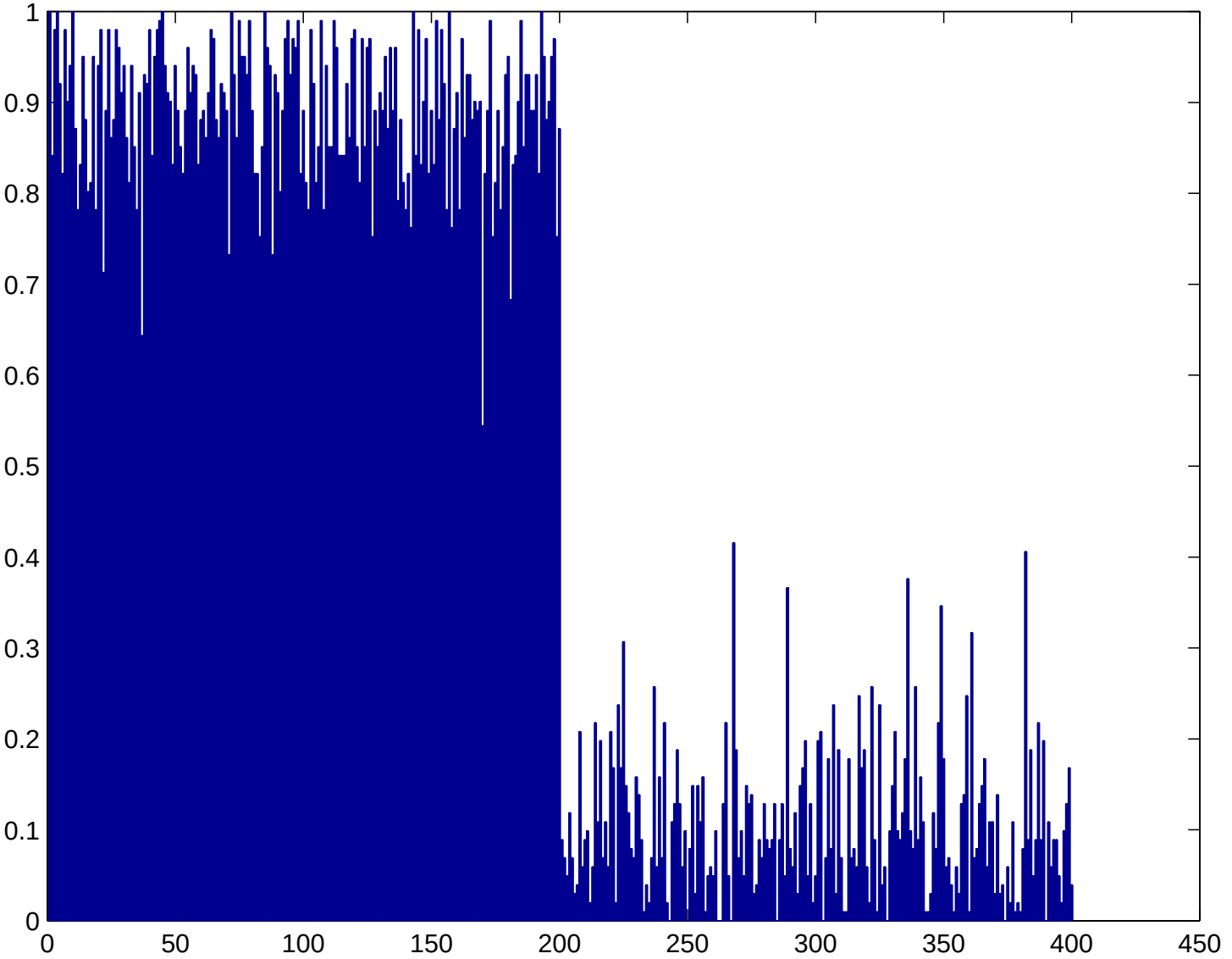


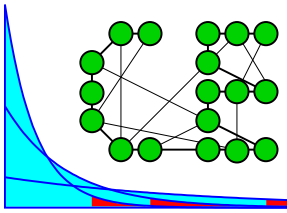
Max-Cut



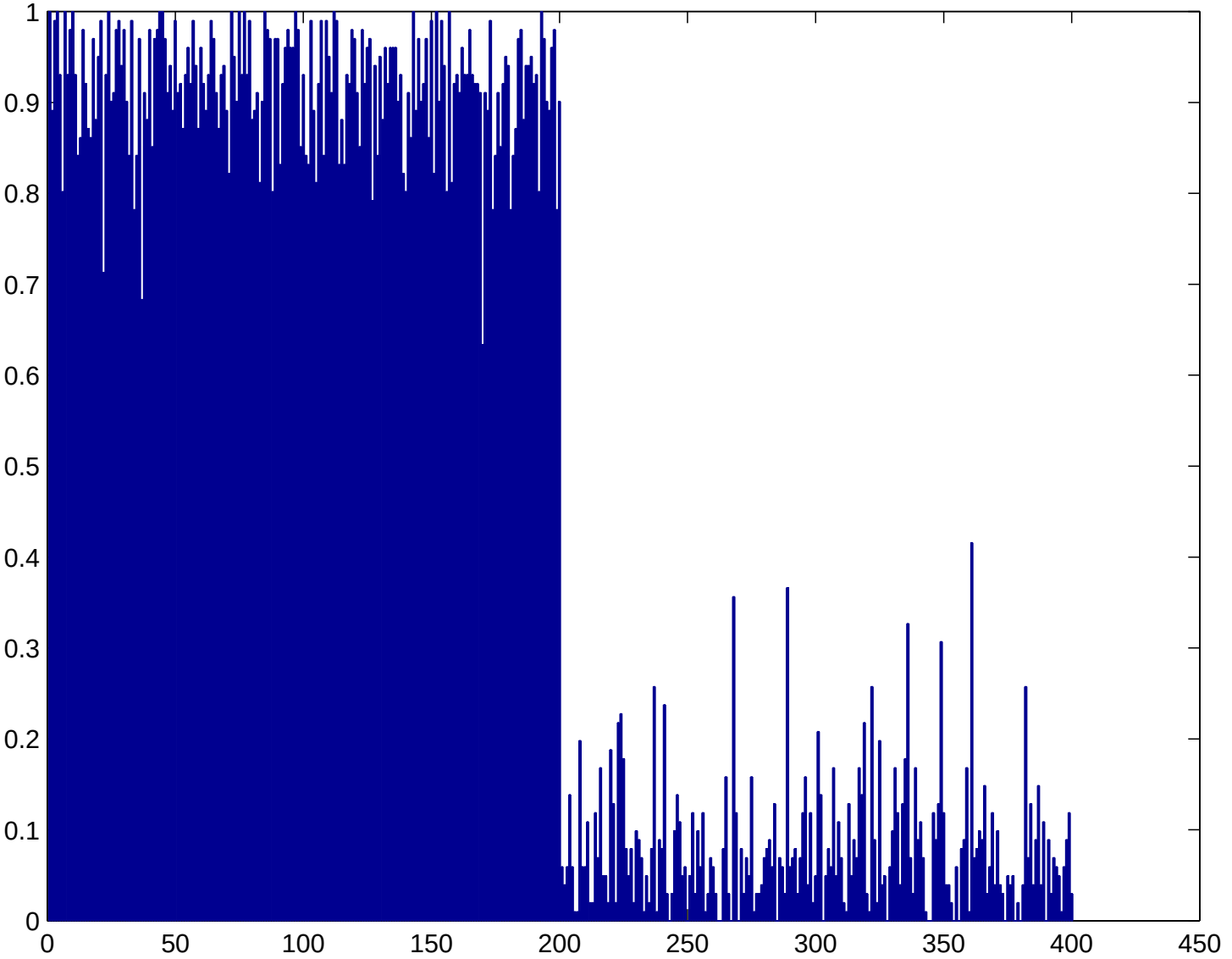


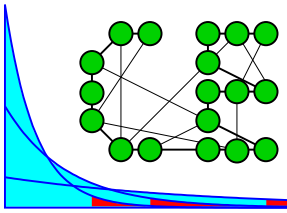
Max-Cut



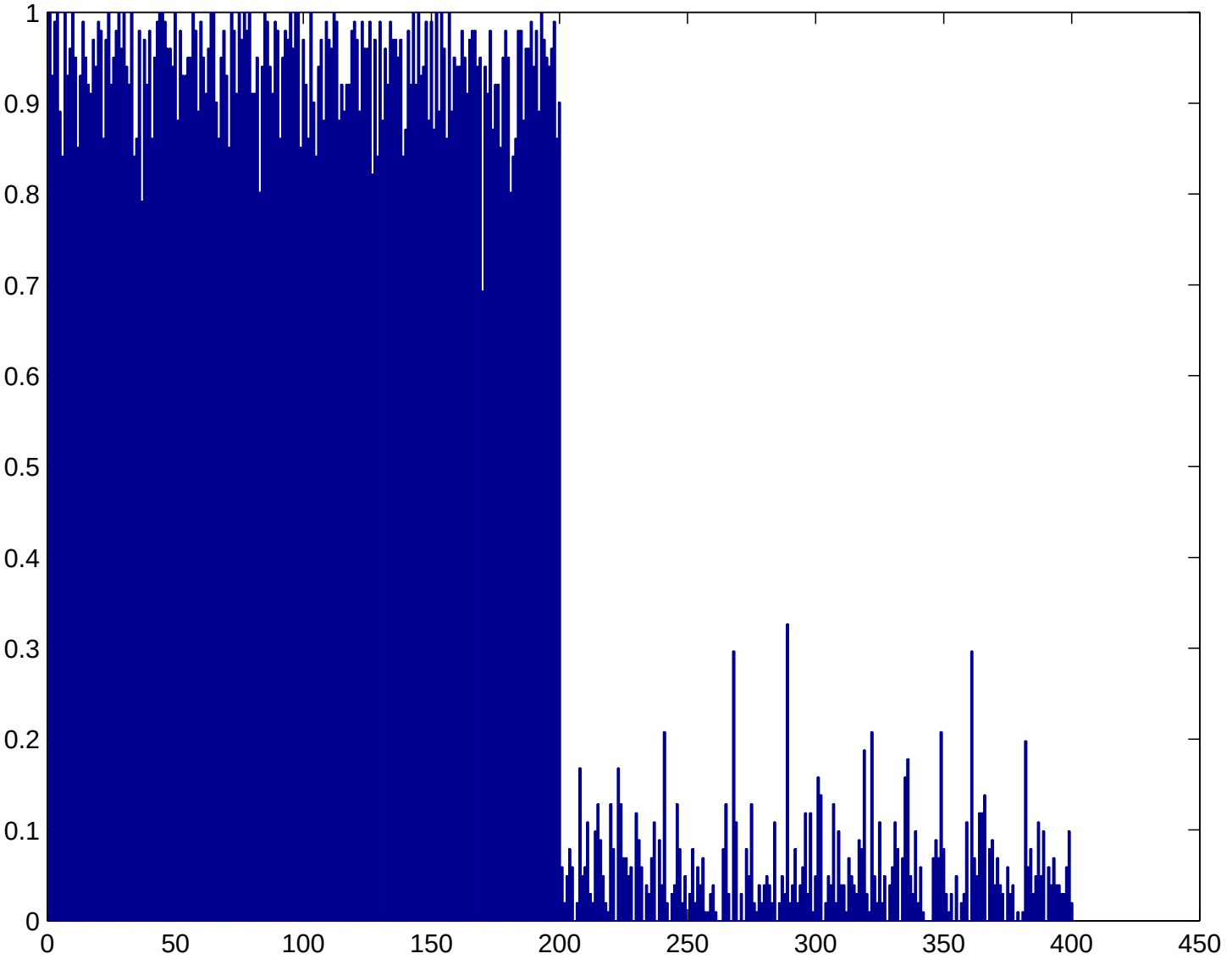


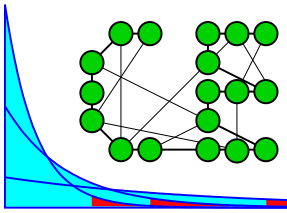
Max-Cut



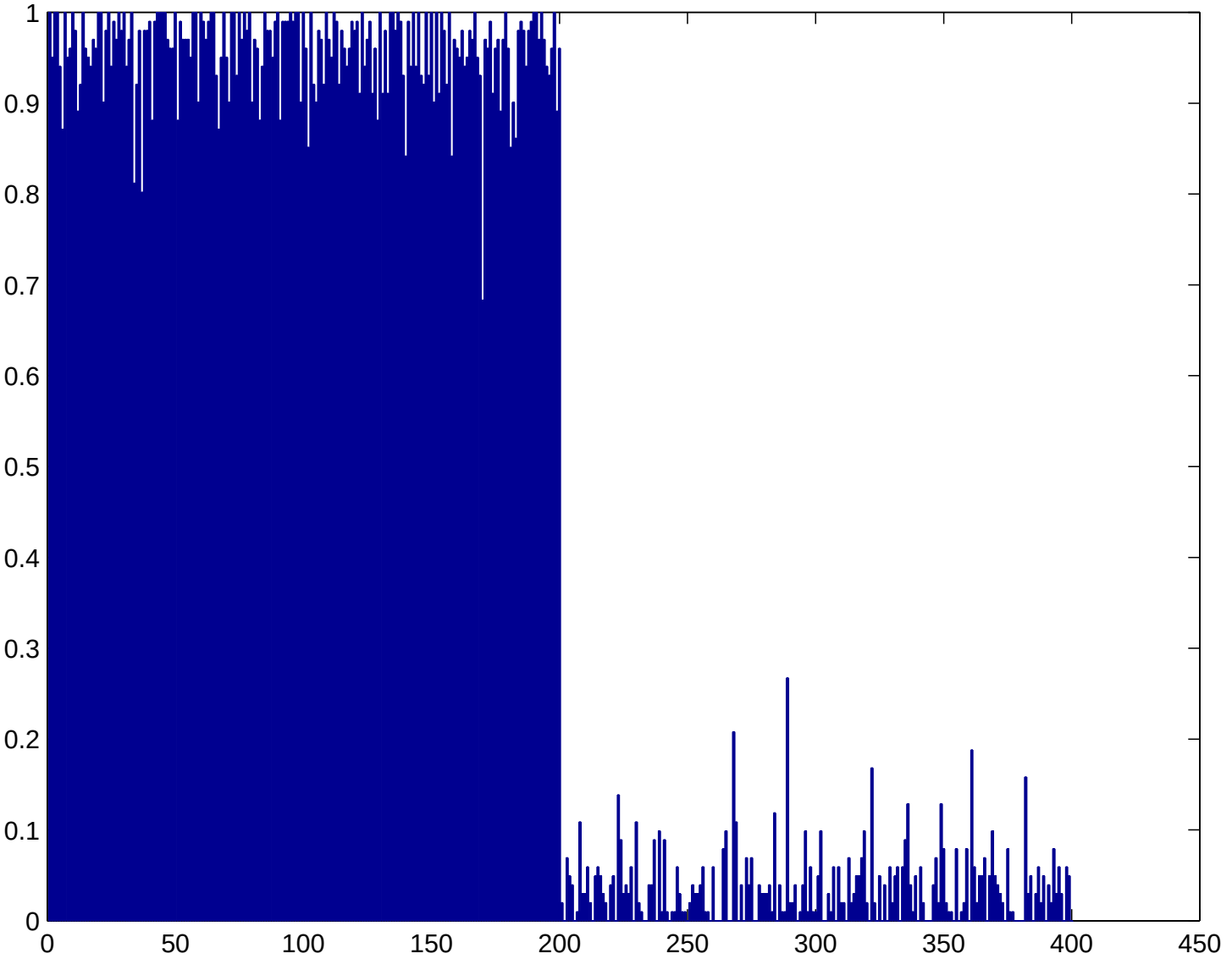


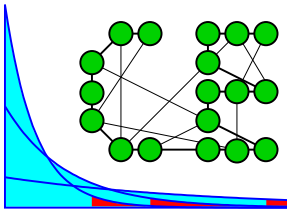
Max-Cut



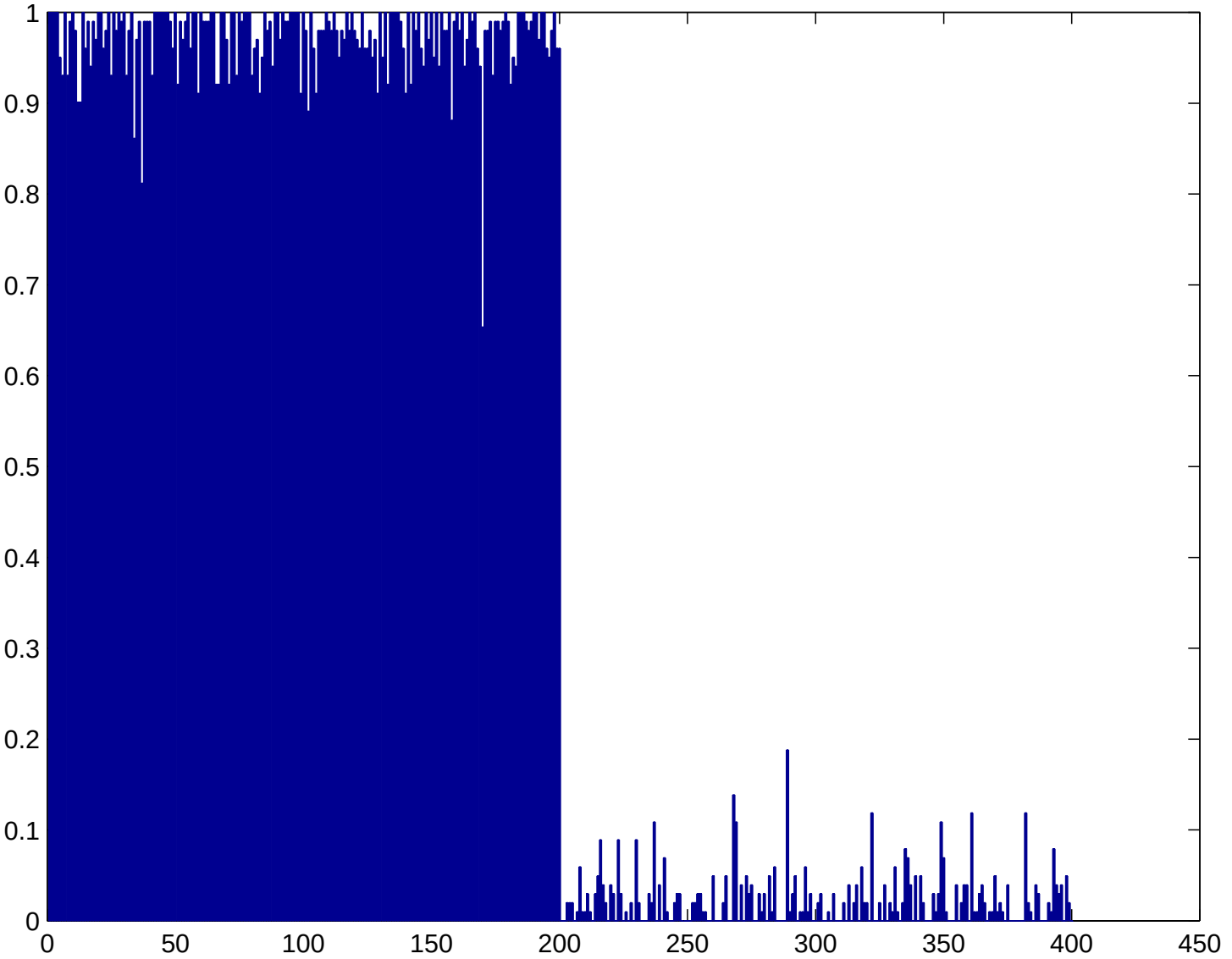


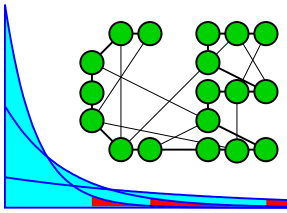
Max-Cut



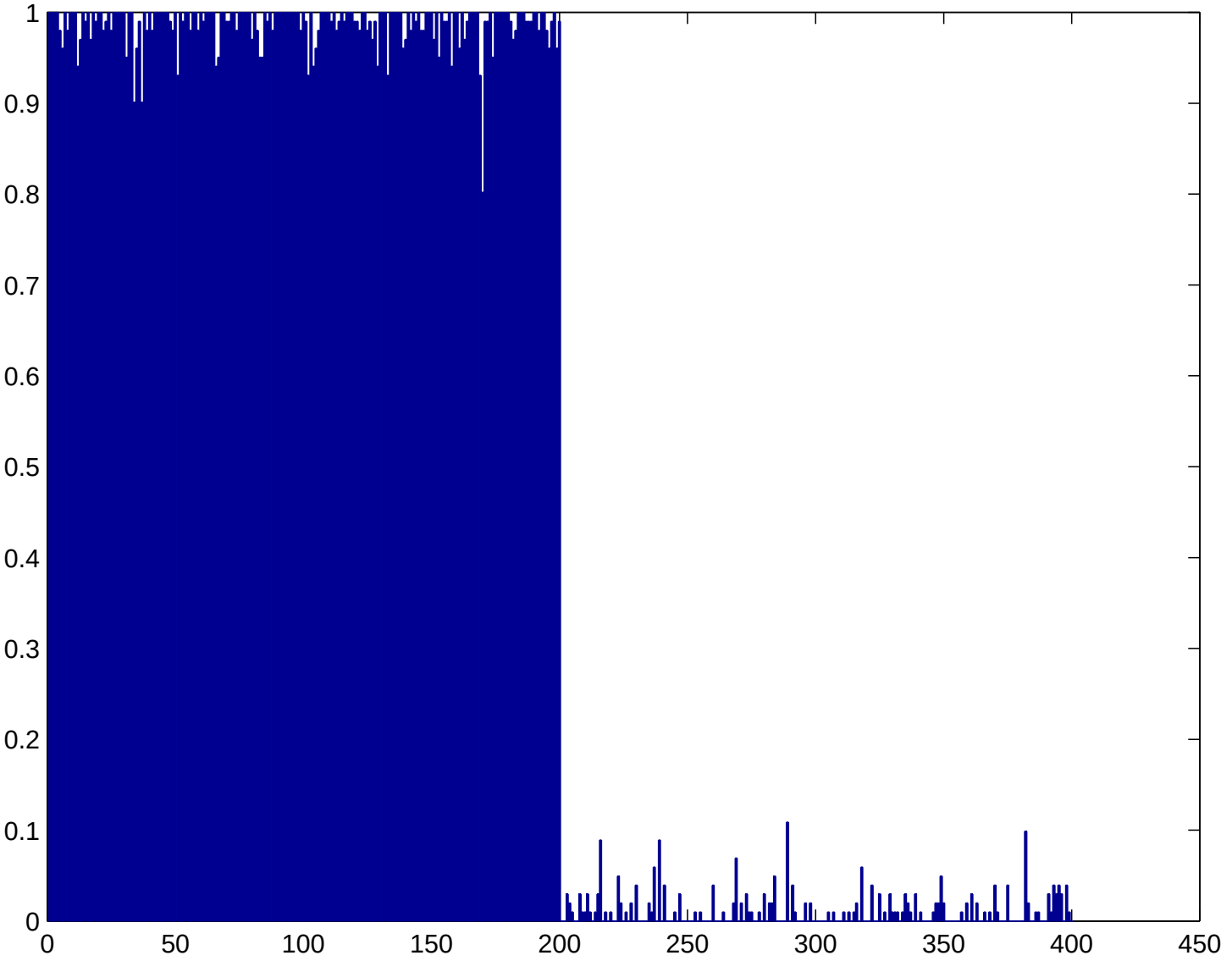


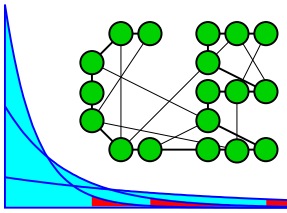
Max-Cut



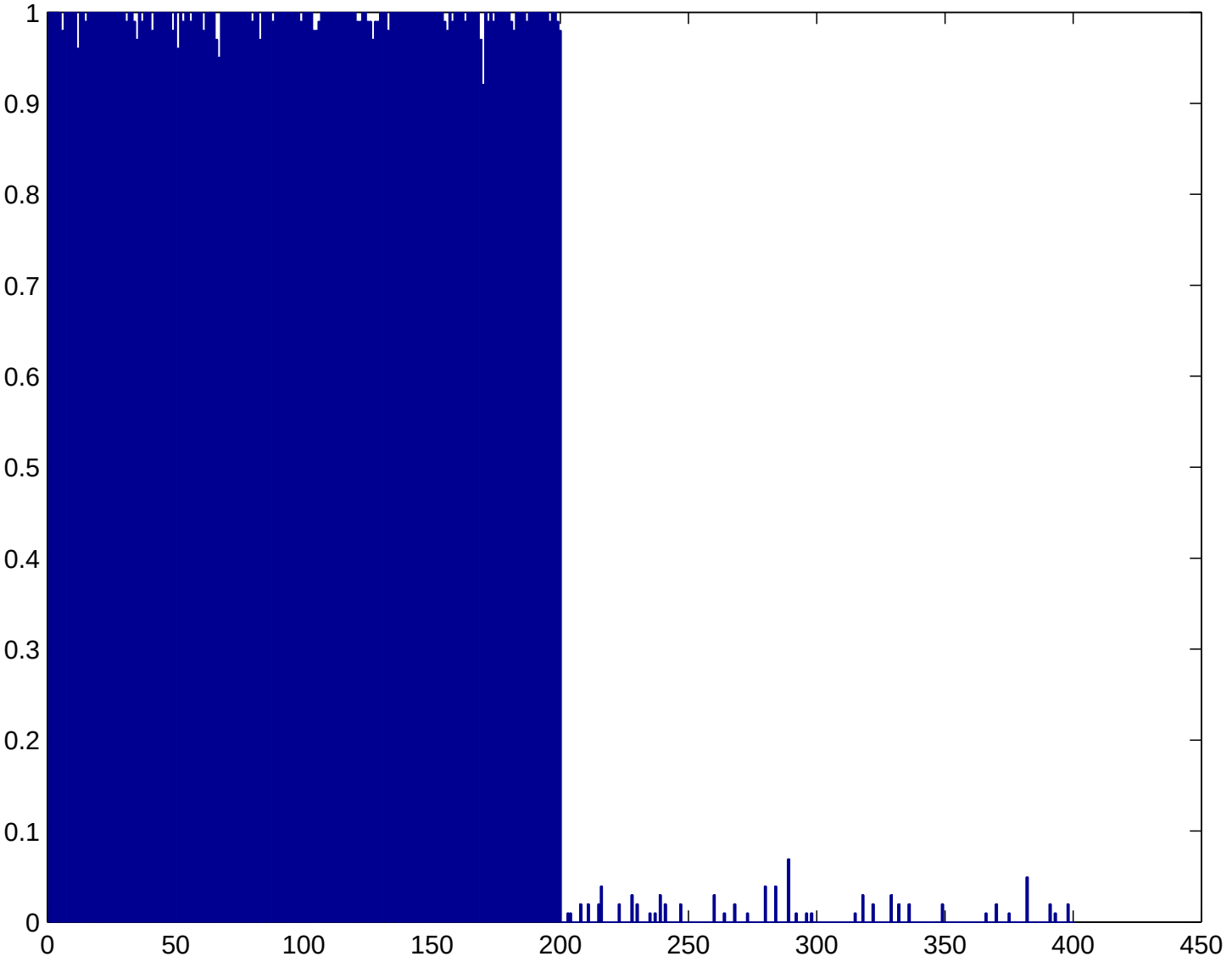


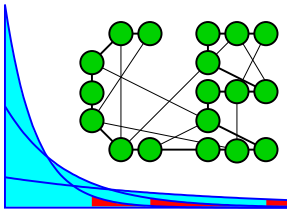
Max-Cut



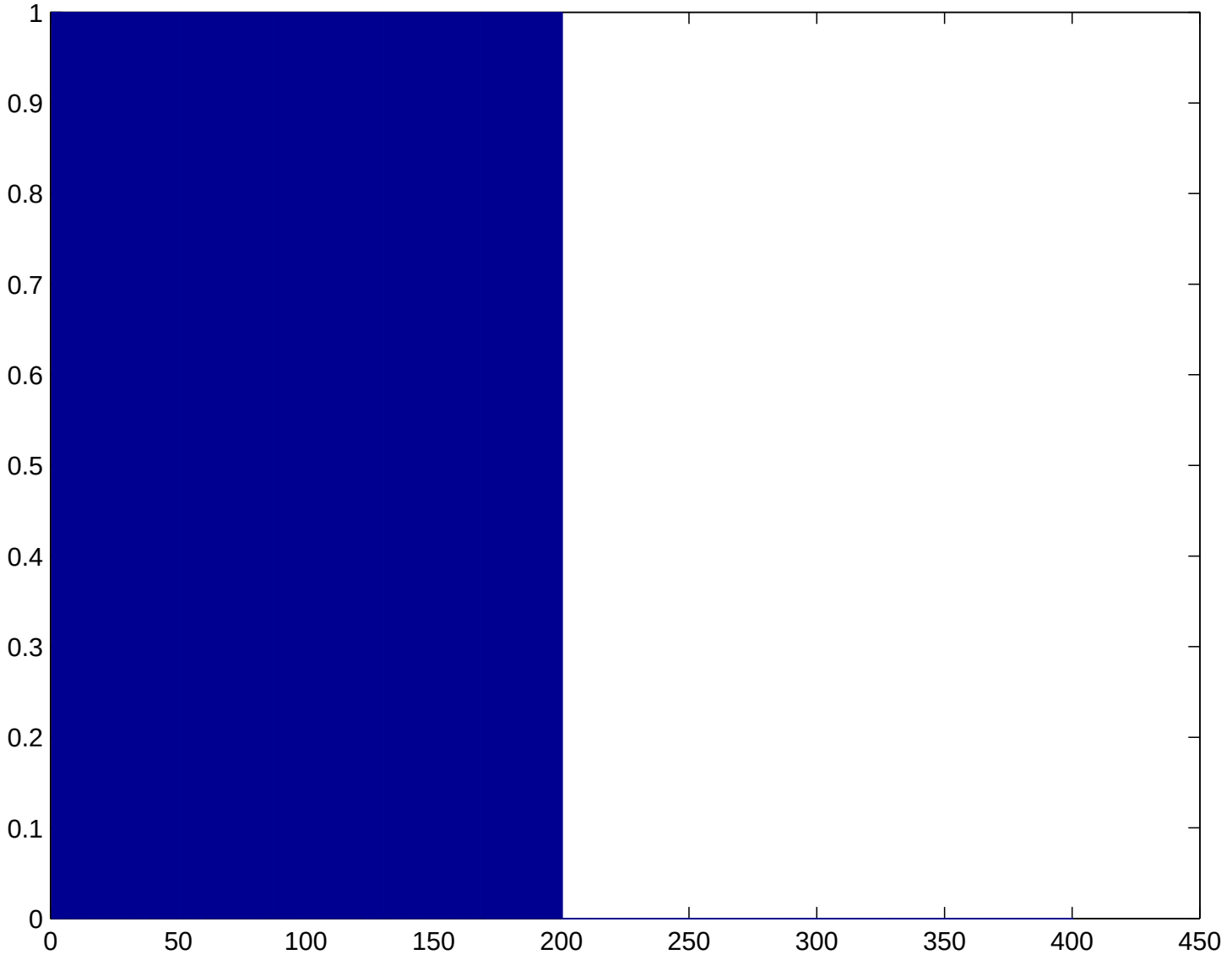


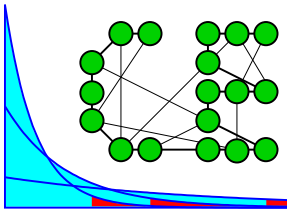
Max-Cut



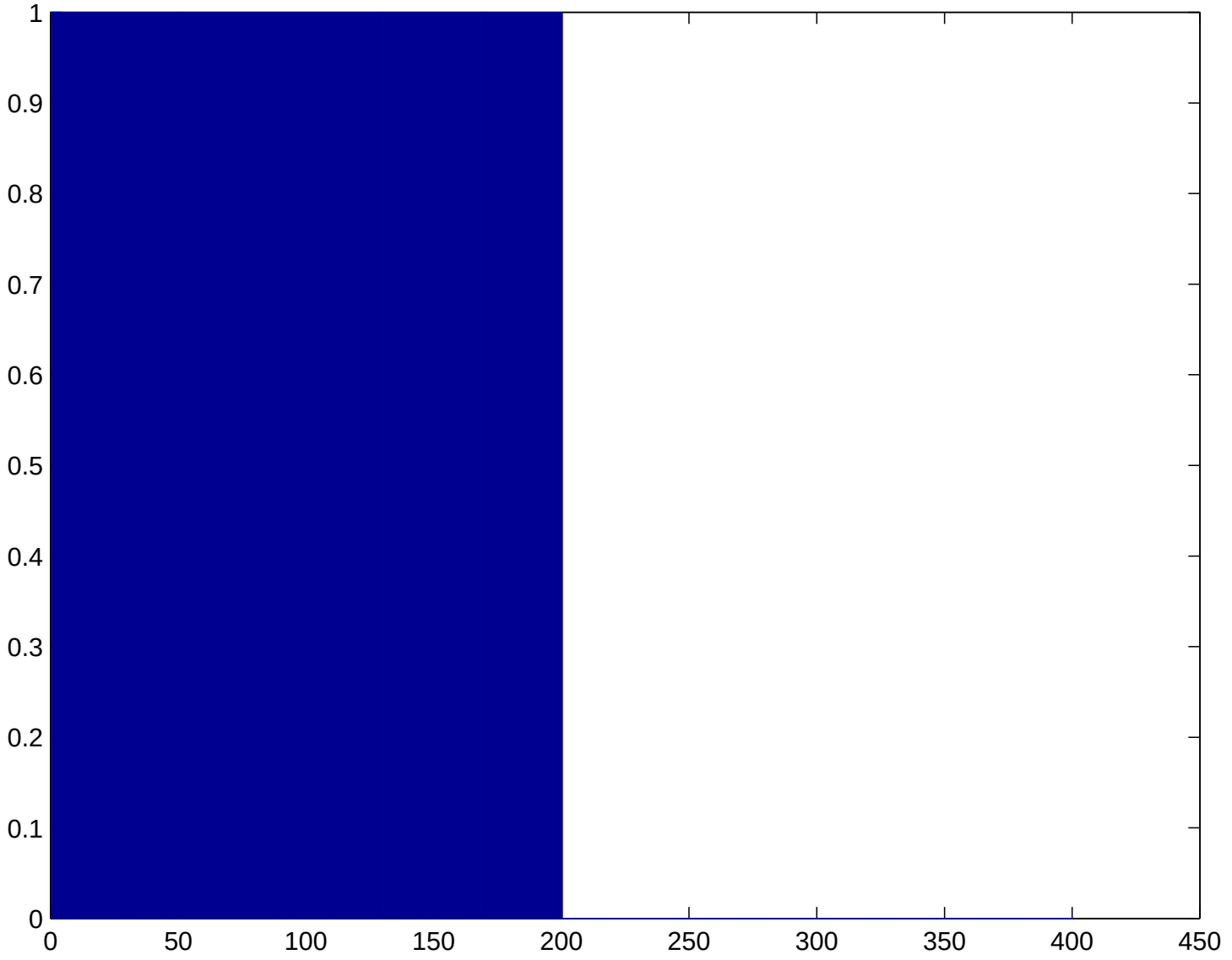


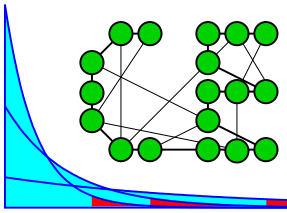
Max-Cut



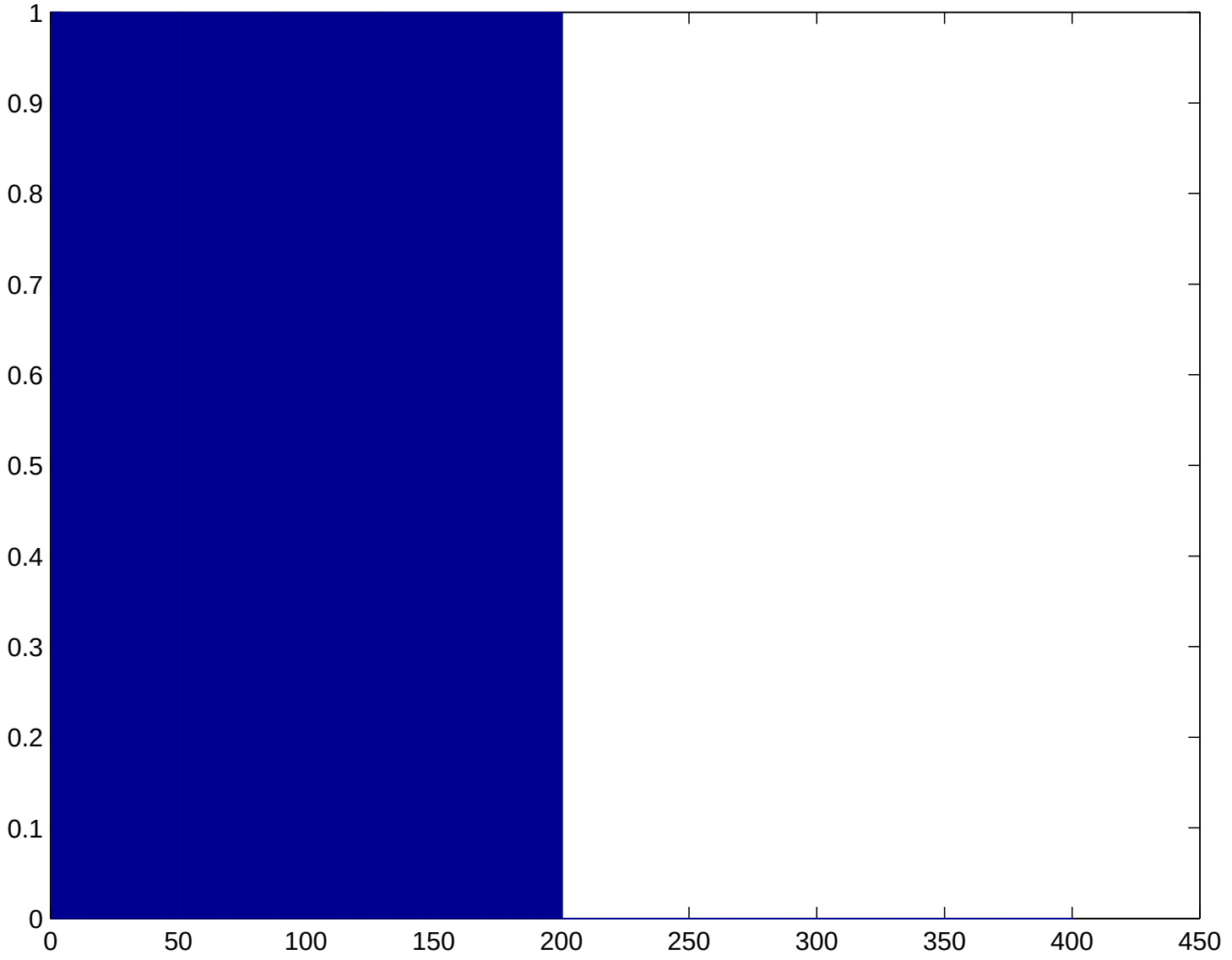


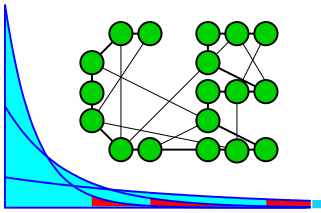
Max-Cut



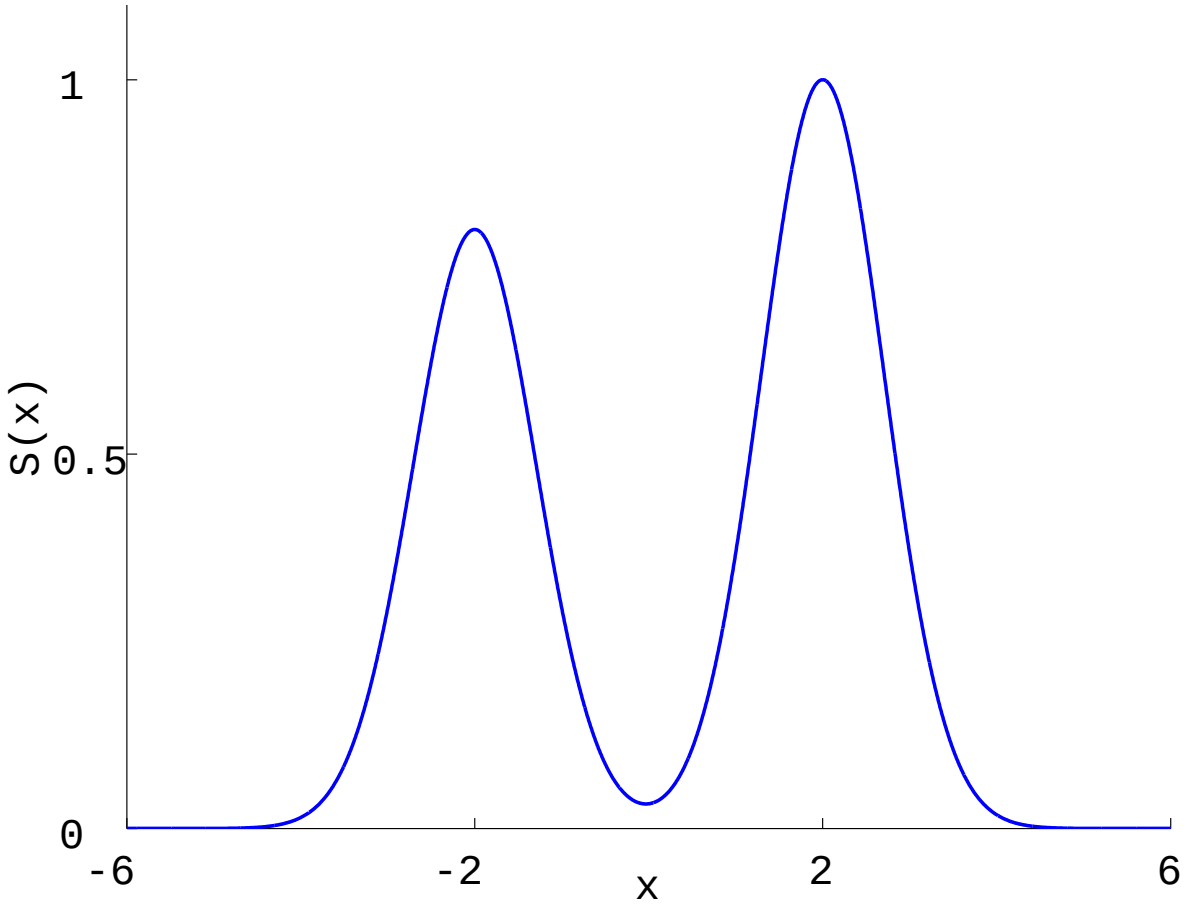


Max-Cut

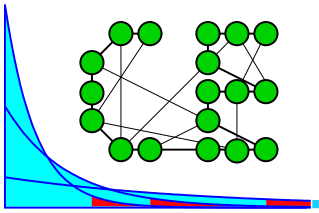




Example: Continuous Optimization

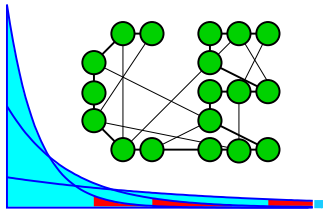


$$S(x) = \mathrm{e}^{-(x-2)^2} + 0.8\,\mathrm{e}^{-(x+2)^2}$$

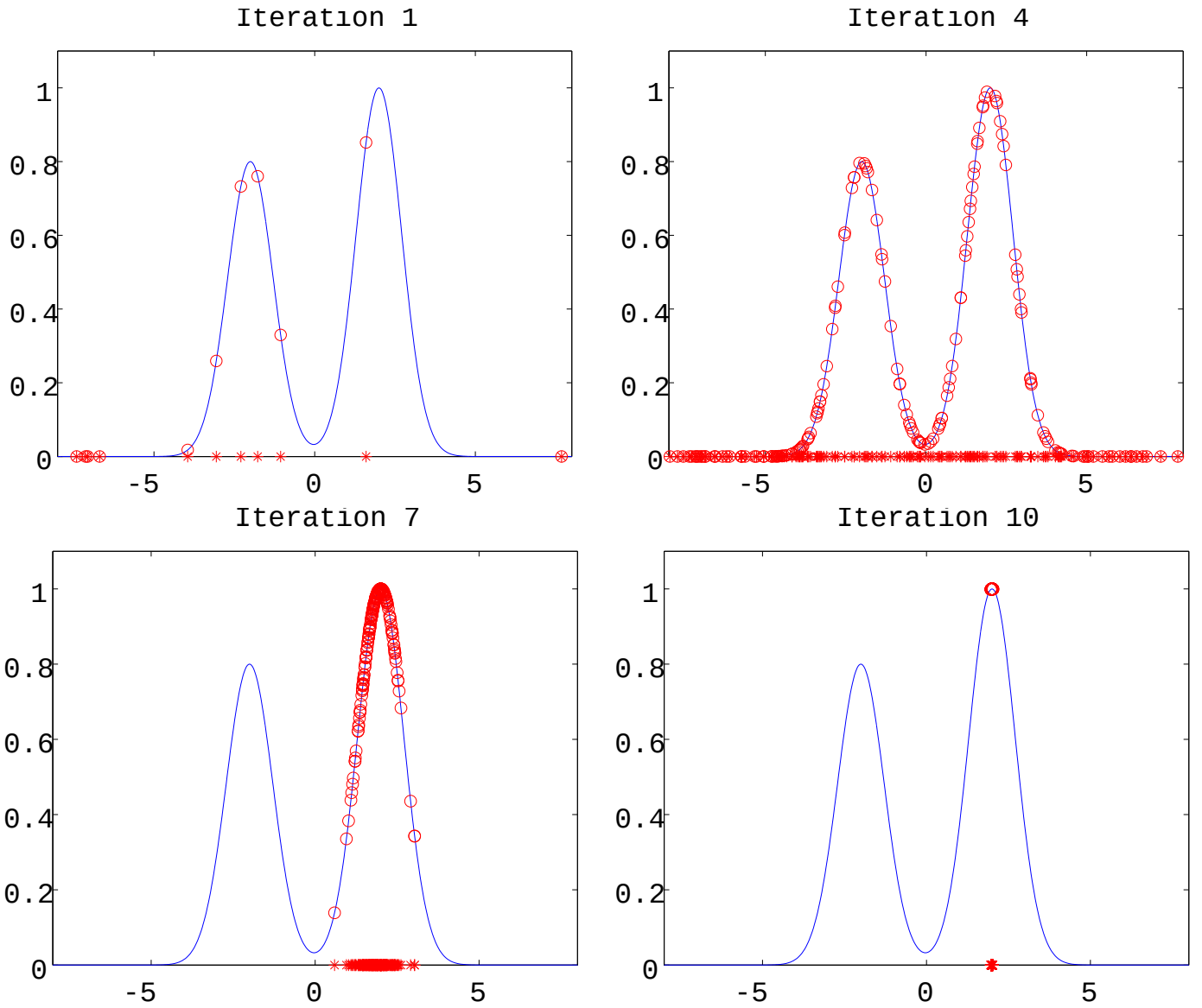


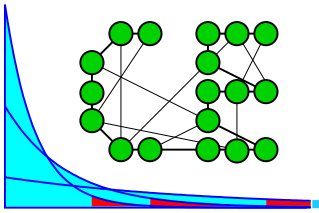
Matlab Program

```
S = inline('exp(-(x-2).^2) + 0.8*exp(-(x+2).^2)');
mu = -10; sigma = 10; rho = 0.1; N = 100; eps = 1E-3;
t=0;  % iteration counter
while sigma > eps
  t = t+1;
  x = mu + sigma*randn(N,1);
  SX = S(x); % Compute the performance.
  sortSX = sortrows([x SX],2);
  mu = mean(sortSX((1-rho)*N:N,1));
  sigma = std(sortSX((1-rho)*N:N,1));
  fprintf('%g  %6.9f  %6.9f  %6.9f \n', t, S(mu),mu, sigma)
end
```



Numerical Result





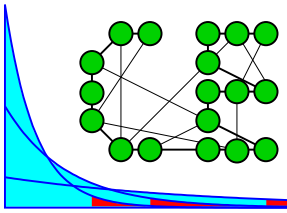
Example: Constrained Optimization

The nonlinear programming problem is to find $\boldsymbol{x}^*$ that minimizes the objective function

$$S(\boldsymbol{x}) = \sum_{j=1}^{10} x_j \left(c_j + \ln \frac{x_j}{x_1 + \cdots + x_{10}} \right),$$

(the c_i are constants) subject to the following set of constraints:

$$\begin{aligned} x_1 + 2x_2 + 2x_3 + x_6 + x_{10} - 2 &= 0, \\ x_4 + 2x_5 + x_6 + x_7 - 1 &= 0, \\ x_3 + x_7 + x_8 + 2x_9 + x_{10} - 1 &= 0, \\ x_j &\geq 0.000001, \quad j = 1, \ldots, 10. \end{aligned}$$



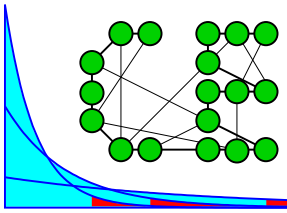
The best known solution in Hock & Schittkowski (Test Examples for Nonlinear Programming Code) was

$$\boldsymbol{x}^* = (0.01773548, 0.08200180, 0.8825646, 0.0007233256, 0.4907851,$$
$$0.0004335469, 0.01727298, 0.007765639, 0.01984929, 0.05269826)$$

with $S(\boldsymbol{x}^*) = -47.707579$. However, using genetic algorithms Michalewicz (Genetic Algorithm + Data Structures = Evolution Programs) finds a better solution:

$$\boldsymbol{x}^* = (0.04034785, 0.15386976, 0.77497089, 0.00167479, 0.48468539,$$
$$0.00068965, 0.02826479, 0.01849179, 0.03849563, 0.10128126),$$

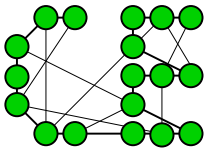
with $S(\boldsymbol{x}^*) = -47.760765$.



Using the CE method we can find an even better solution (in less time):

$$\boldsymbol{x}^* = (0.04067247, 0.14765159, 0.78323637, 0.00141368, 0.48526222,$$
$$0.00069291, 0.02736897, 0.01794290, 0.03729653, 0.09685870)$$

and $S(\boldsymbol{x}^*) = -47.76109081$ (using $\varepsilon = 10^{-4}$).



How was it done?

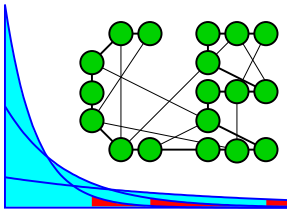
The first step is to reduce the search space by expressing $x_1,\ x_4,\ x_8$ in terms of the other seven variables:

$$\begin{aligned} x_1 &= 2-(2x_2+2x_3+x_6+x_{10}), \\ x_4 &= 1-(2x_5+x_6+x_7), \\ x_8 &= 1-(x_3+x_7+2x_9+x_{10}). \end{aligned}$$

Hence, we have reduced the original problem of ten variables to that of a function of seven variables, which are subject to the constraints $x_2, x_3, x_5, x_6, x_7, x_9, x_{10} \geq 0.000001$, and

$$2-(2x_2+2x_3+x_6+x_{10}) \quad \geq \quad 0.000001,$$

etc.

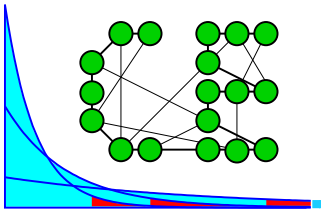


Next step choose an appropriate rectangular "trust region" $\mathcal{R}$ that contains the optimal solution.

Draw the samples from a truncated normal distribution (with independent components) on this space $\mathcal{R}$. Reject the sample if the constraints are not satisfied (acceptance–rejection method).

The updating formulas remain the same as for untruncated normal sampling: update via the mean and variance of the elite samples.

An alternative is to add a penalty term to S.



Cross-Entropy: Some Theory

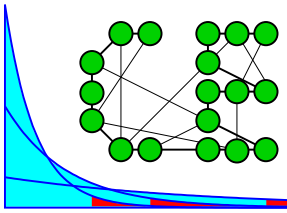
Let $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ be a random sample from $f(\cdot; \boldsymbol{v}_{t-1})$.
Let γ_t be the $(1-\rho)$ quantile of the performance:

$$\rho = \mathbb{P}_{\boldsymbol{v}_{t-1}}(S(\boldsymbol{X}) \geq \gamma_t), \quad \text{with } \boldsymbol{X} \sim f(\cdot; \boldsymbol{v}_{t-1}).$$

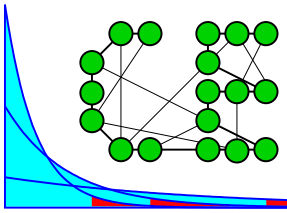
Consider now the rare event estimation problem where we estimate ρ via Importance Sampling:

$$\hat{\rho} = \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \gamma_t\}} \frac{f(\boldsymbol{X}_i; \boldsymbol{v}_{t-1})}{g(\boldsymbol{X}_i),} \quad \text{with } \boldsymbol{X}_1, \ldots, \boldsymbol{X}_N \sim g(\cdot).$$

A zero variance estimator is $g^*(\boldsymbol{x}) := \frac{I_{\{S(\boldsymbol{x}) \geq \gamma_t\}} f(\boldsymbol{x}; \boldsymbol{v}_{t-1})}{\rho}$.
Problem: g^* depends on the unknown ρ.



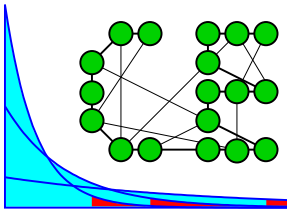
Idea: Update $\boldsymbol{v}_t$ such that the "distance" between the densities g^* and $f(\cdot;\boldsymbol{v}_t)$ is minimal.



Idea: Update $\boldsymbol{v}_t$ such that the "distance" between the densities g^* and $f(\cdot;\boldsymbol{v}_t)$ is minimal.

The **Kullback-Leibler** or **cross-entropy** distance is defined as:

$$\begin{aligned}\mathcal{D}(g,h) &= \mathbb{E}_g \log \frac{g(\boldsymbol{X})}{h(\boldsymbol{X})} \\ &= \int g(\boldsymbol{x}) \log g(\boldsymbol{x})\, d\boldsymbol{x} - \int g(\boldsymbol{x}) \log h(\boldsymbol{x})\, d\boldsymbol{x}\ .\end{aligned}$$

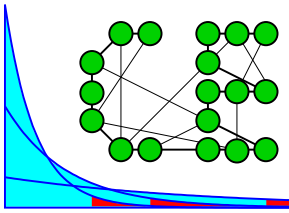


Idea: Update $\boldsymbol{v}_t$ such that the "distance" between the densities g^* and $f(\cdot;\boldsymbol{v}_t)$ is minimal.

The **Kullback-Leibler** or **cross-entropy** distance is defined as:

$$\begin{aligned}\mathcal{D}(g,h) &= \mathbb{E}_g \log \frac{g(\boldsymbol{X})}{h(\boldsymbol{X})} \\ &= \int g(\boldsymbol{x}) \log g(\boldsymbol{x})\, d\boldsymbol{x} - \int g(\boldsymbol{x}) \log h(\boldsymbol{x})\, d\boldsymbol{x}\ .\end{aligned}$$

Determine the optimal $\boldsymbol{v}_t$ from $\min_{\boldsymbol{v}} \mathcal{D}(g^*, f(\cdot;\boldsymbol{v}))$.



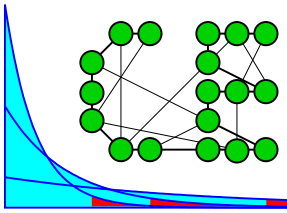
Idea: Update $\boldsymbol{v}_t$ such that the "distance" between the densities g^* and $f(\cdot;\boldsymbol{v}_t)$ is minimal.

The **Kullback-Leibler** or **cross-entropy** distance is defined as:

$$\begin{aligned}\mathcal{D}(g,h) &= \mathbb{E}_g \log \frac{g(\boldsymbol{X})}{h(\boldsymbol{X})} \\ &= \int g(\boldsymbol{x}) \log g(\boldsymbol{x})\, d\boldsymbol{x} - \int g(\boldsymbol{x}) \log h(\boldsymbol{x})\, d\boldsymbol{x} \ .\end{aligned}$$

Determine the optimal $\boldsymbol{v}_t$ from $\min_{\boldsymbol{v}} \mathcal{D}(g^*, f(\cdot;\boldsymbol{v}))$.

This is equivalent to solving $\max_{\boldsymbol{v}} \mathbb{E}_{\boldsymbol{v}_{t-1}} I_{\{S(\boldsymbol{X})\geq\gamma_t\}} \log f(\boldsymbol{X};\boldsymbol{v})$.



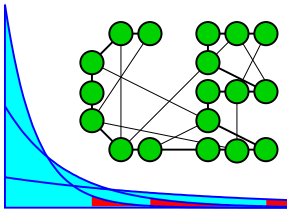
We may *estimate* the optimal solution $\boldsymbol{v}_t$ by solving the following stochastic counterpart:

$$\max_{\boldsymbol{v}} \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \gamma_t\}} \log f(\boldsymbol{X}_i; \boldsymbol{v}) \ ,$$

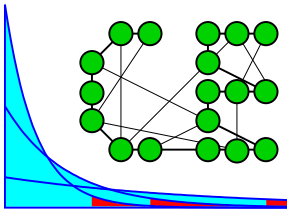
where $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ is a random sample from $f(\cdot; \boldsymbol{v}_{t-1})$.
Alternatively, solve:

$$\frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \gamma_t\}} \nabla \log f(\boldsymbol{X}_i; \boldsymbol{v}) = \mathbf{0},$$

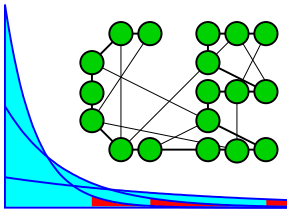
where the gradient is with respect to $\boldsymbol{v}$.



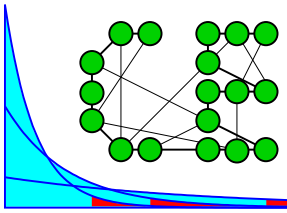
- The solution to the CE program can often be calculated analytically (Maximum Likelihood Estimator!).



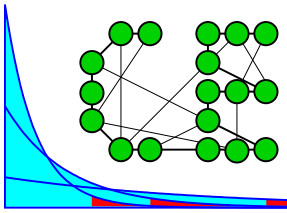
- The solution to the CE program can often be calculated analytically (Maximum Likelihood Estimator!).
- Note that the CE program is useful only when under $\boldsymbol{v}_{t-1}$ the event $\{S(\boldsymbol{X}) \geq \gamma_t\}$ is not too rare, say $\rho \geq 10^{-3}$.



- The solution to the CE program can often be calculated analytically (Maximum Likelihood Estimator!).
- Note that the CE program is useful only when under $\boldsymbol{v}_{t-1}$ the event $\{S(\boldsymbol{X}) \geq \gamma_t\}$ is not too rare, say $\rho \geq 10^{-3}$.

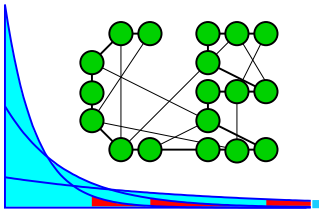


- The solution to the CE program can often be calculated analytically (Maximum Likelihood Estimator!).
- Note that the CE program is useful only when under $\boldsymbol{v}_{t-1}$ the event $\{S(\boldsymbol{X}) \geq \gamma_t\}$ is not too rare, say $\rho \geq 10^{-3}$.



- The solution to the CE program can often be calculated analytically (Maximum Likelihood Estimator!).
- Note that the CE program is useful only when under $\boldsymbol{v}_{t-1}$ the event $\{S(\boldsymbol{X}) \geq \gamma_t\}$ is not too rare, say $\rho \geq 10^{-3}$.

By iterating over t we obtain a sequence of reference parameters $\{\boldsymbol{v}_t,\, t \geq 0\} \to \boldsymbol{v}^*$ and a sequence of levels $\{\gamma_t,\, t \geq 1\} \to \gamma^*$.



Updating Example

Suppose X has iid exponential components:

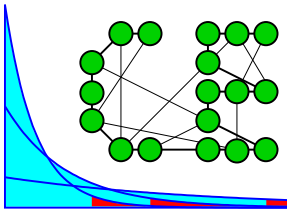
$$f(\boldsymbol{x};\boldsymbol{v}) = \exp\left(-\sum_{j=1}^{5}\frac{x_j}{v_j}\right)\prod_{j=1}^{5}\frac{1}{v_j}\ .$$

The optimal $\boldsymbol{v}$ follows from the system of equations

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

Since

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



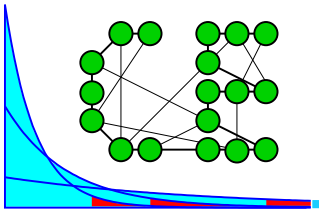
we have for the jth equation

$$\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i)\geq\gamma\}} \left(\frac{X_{ij}}{v_j^2} - \frac{1}{v_j}\right) = 0 \ ,$$

whence,

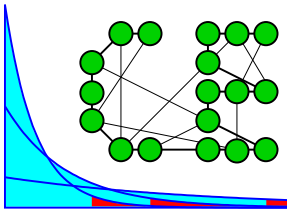
$$v_j = \frac{\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i)\geq\gamma\}} X_{ij}}{\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i)\geq\gamma\}}} \ ,$$

Thus the updating rule is here to take the mean of the elite samples (MLE).



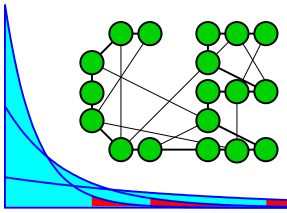
Conclusions

- The CE method optimizes the sampling distribution.
- Advantages:
  - Universally applicable (discrete/continuous/mixed problems)
  - Very easy to implement
  - Easy to adapt, e.g. when constraints are added
  - Works generally well
- Disadvantages:
  - It is a generic method
  - Performance function should be relatively cheap
  - Tweaking (modifications) may be required



Discussion + Further Research

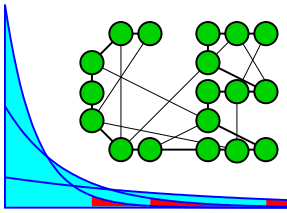
- Synergies with related fields need to be further developed:
  - Evolutionary Algorithms
  - Stochastic Approximation
  - Simulated Annealing
  - Probability Collectives
  - Reinforcement Learning
  - Bayesian Inference
  - (Multi-actor) game theory



Discussion + Further Research

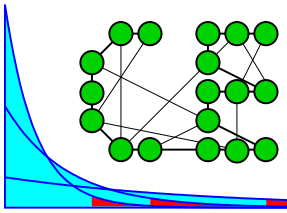
- Convergence properties need to be further examined:
  - Continuous optimization (e.g., Thomas Taimre's thesis)
  - Rare event simulation (Homem-de-Melo, Margolin)
  - Significance of various modifications
  - Role of the sampling distribution
- Noisy optimization: very little needs to be changed!
- Test problems and "best" code need to be made available:

  CE Toolbox: `www.maths.uq.edu.au/CEToolBox`



Acknowledgements

- Zdravko Botev, Jenny Liu, Sho Nariai, Thomas Taimre
- Phil Pollett



Acknowledgements

- Zdravko Botev, Jenny Liu, Sho Nariai, Thomas Taimre
- Phil Pollett

Thank You!