# Lecture notes on Monte Carlo simulations, Stochastic Processes, MCMC, Simulated Annealing

Carlo Lucibello

18th September, 2020

**Abstract**

In this lecture notes for undergrad students we present the basic Monte Carlo techniques used for effectively dealing with a variety of different problems: approximating volumes and integrals in high dimensional spaces, sampling from intractable probability distributions (using the Metropolis-Hasting algorithm) and solving hard combinatorial optimization problems (with Simulated Annealing).

## Contents

# 1 Monte Carlo Methods

The family of Monte Carlo methods provide approximate solutions to a variety of mathematical problems by performing statistical sampling experiments on a computer [1]. The name was coined by J. von Neumann, while he was working on nuclear fission at Los Alamos laboratories. It was chosen in reference to the city famous for its casino, since the method uses a sequence of random numbers. We will see how Monte Carlo methods can be used for different purposes, such as counting, sampling from intractable probability distributions and solving optimization problems.

## 1.1 Computing volumes with random numbers

A nice simple example to understand the power of Monte Carlo (MC) methods is to use random number generators to compute volumes. This may seem like a fancy way of doing a simple task, but we will see that even in trivial cases like this we will achieve interesting results in a very straightforward way. At the end of this section we will extend the ideas behind approximating volumes by MC to evaluation of high dimensional integrals.

As a warm-up example we will estimate the value of $\pi$. Consider the following picture:

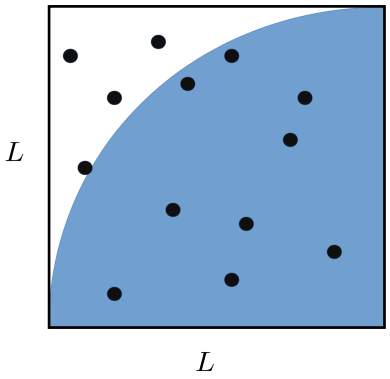


**Figure 1**

We know that the area of the black-bordered square ("total area") is equal to $L^2$ and that the "blue area" is equal to $\frac{\pi}{4}L^2$, thus we can obtain the value of $\pi$ as

$$\pi = 4 \times \frac{\text{blue area}}{\text{total area}} \tag{1}$$

Now we just need to computes the two areas, or more precisely, their ratio. The ratio can be estimated by generating random points uniformly distributed in the square and counting the ones that fell in the blue are:

$$\pi \approx 4 \times \frac{\text{number of points in blue area}}{\text{total number of points}} \tag{2}$$

The same idea can be applied to every shape that we put inside a box of know dimensions. Take for example the following shape of a lake:

[1] G.S. Fishman, "Monte Carlo: concepts, algorithms and applications", Springer, 1996

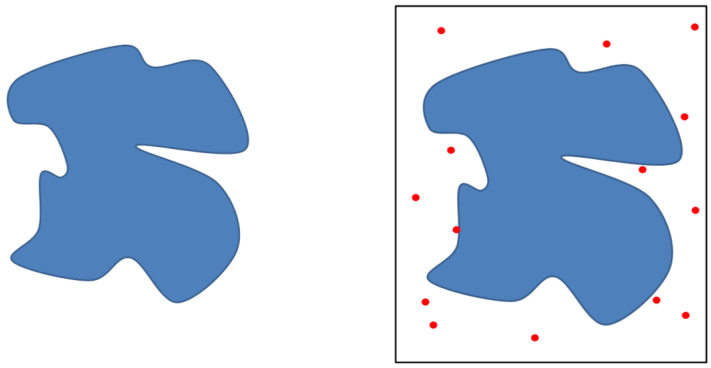

**Figure 2**

How to estimate the area of this lake? Just put the lake in a framework of known area, then generate random points within it, then count how many of them are in the lake.

From these simple examples it is apparent that the resulting estimate is reliable only when:

- samples are uniformly distributed;
- a sufficiently large number of samples is generated.

Therefore, two main questions arise:

- how to generate uniformly distributed random numbers?
- what is the relationship between the number of samples and the accuracy of the estimate?

### 1.1.1 Pseudo-random Numbers

Ideally, we would need unlimited sequences of random numbers generated according to a probability distribution with continuous domain. In practice, computers are:

- Discrete → Finite precision
- Finite → Finite domain and limited sequence length
- Deterministic → Sequences can not be truly random

In particular, since computer generated sequences are deterministic, we call the seemingly random numbers produced by a computer *pseudo-random numbers*. Many software packages include algorithms for generating sequences of pseudo-random numbers. These sequences are almost indistinguishable from sequences of truly random numbers, according to some statistical standards. Generally, an algorithm for pseudo-random number generation can be described as follows: let $z_m$ be the $m$-th number of the sequence we want to generate, then $z_m$ will be produced according to the (deterministic) rule

$$z_m = f\left(A, z_{m-1}, ..., z_{m-p}\right) \tag{3}$$

where $f$ is a function characteristic of the specific generator, $A$ is a set of parameters and $z_{m-1}, ..., z_{m-p}$ are the previous $p$ numbers in the sequence. We can state a number of general properties of pseudo-random number generators:

- they generate a sequence of numbers from a finite set of integers;
- each number is generated as a function of the last $p$ numbers in the sequence;
- the function $f$ has the topological transitivity property: every subsequence is produced (early or later) by repeatedly computing $f$, starting from any subsequence of length $p$;
- the set of parameters $A$ is chosen in order to maximize the length of the sequence before it starts repeating.

One would want any subsequence of length $n$ to correspond to a random number uniformly and independently distributed in the unit hypercube. Using fast generators, this characterization is less and less satisfied by pseudo-random sequences as $n$ increases. While the error in the estimate obtained with Monte Carlo methods decreases with the number of samples, the error induced by pseudo-random numbers generation does not. This puts a bound to the precision that can be achieved by MC methods.

Common pseudo-random number generators used in practice are variation of linear congruential generators. In particular, `numpy` currently uses a generator called PCG64. Another class of generators which is very well known for their good sampling properties is that of Mersenne Twister engines.

### 1.1.2 Sample size and accuracy

In this section we will understand the main advantage of the Monte Carlo method, namely that the error on volumes estimates decreases as $n^{1/2}$, whereas using other (approximated) methods based on the evaluation of $n$ points in $m$ dimensions the error decreases as $n^{1/m}$, therefore much more slowly when $m > 2$.

We consider two related problems:

- (approximately) computing integrals (areas, volumes,...);
- (approximately) counting.

The former problem arises with continuous functions, the latter one with discrete sets. The Monte Carlo method provides

- a point estimate, i.e. an approximate value for the volume;
- a confidence interval, representing how reliable the estimate is (Chap. 2 Fisherman).

The number of sample points we use to compute the approximation is called *sample size*. Given a desired accuracy requirement, we would like to know the corresponding minimum sample size. We will see that the probabilistic analysis of the Monte Carlo provides such estimate. Additionally, efficiency-improving techniques have been devised to reduce the sample size and hence the computing time required by the method.

Given a continuous function in an $m$-dimensional space, we want to (approximately) compute an area or a volume enclosed by the function. Let $R$ denote a region of unknown volume $\lambda(R)$ in the $m$-dimensional unit hypercube $[0,1]^m$. Now assume that we have two functions:

- the function GeneratePoint $(m)$ generates a point in the $m$-dimensional unit hypercube. For the time being we don't specify how the function generates the point and we also allow it to generate a new point based on the previously generated points.
- a boolean function IsInside $(\boldsymbol{x}, R)$ tests whether any given point $\boldsymbol{x}$ is contained in the region $R$ or not (here the bold font denotes a vector).

Then, we can approximate $\lambda(R)$ using the following MC algorithm:

**Algorithm 1** Estimating $\lambda(R)$.

**Require:** $R \subseteq [0,1]^m, n$
**Ensure:** $\bar{\lambda}(R) \approx \lambda(R)$
  $S \leftarrow 0$
  **for** $j \leftarrow 1, ..., n$ **do**
    $\mathbf{x} \leftarrow$ GeneratePoint(m) ▷ $m$-dimensional vector
    **if** IsInside($\mathbf{x}, R$) **then**
      $S \leftarrow S + 1$
    **end if**
  **end for**
  **return** $S/n$

To appreciate the advantage given by the Monte Carlo method, let's now analyze two possible functions GeneratePoint:

1. Each call to GeneratePoint generates a different point on a grid of $n$ points in the unit hypercube
2. The function GeneratePoint generates points drawn from a semi-random sequence, namely a call to GeneratePoint ($m$) generates a point approximately uniformly distributed in $[0,1]^m$ and independent from all other generated points.

### Computing Volumes By Using a Grid

Let's first think in $m = 2$ dimensions: we set $n = 100$ and divide the unit hypercube into $10 \times 10$ smaller identical hypercubes, whose centers form a lattice. We say that the *spacing* of the lattice is $k = 1/10$.

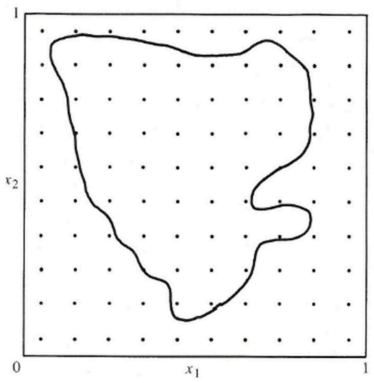


**Figure 3**

In general, using a spacing $k$ in $m$ dimensions requires $n = 1/k^m$ points in the lattice; therefore the spacing can be written as

$$k = \frac{1}{n^{\frac{1}{m}}} \tag{4}$$

The estimate of the volume is given by (remember that the total volume of the hypercube is 1)

$$\lambda_{\text{grid}} = \frac{\text{n. points inside } R}{n} = k^m \times (\text{n. points inside } R) \tag{5}$$

In the following left picture the estimation of the area is the shaded region. The error of the estimate is the dark area in the picture on the right.

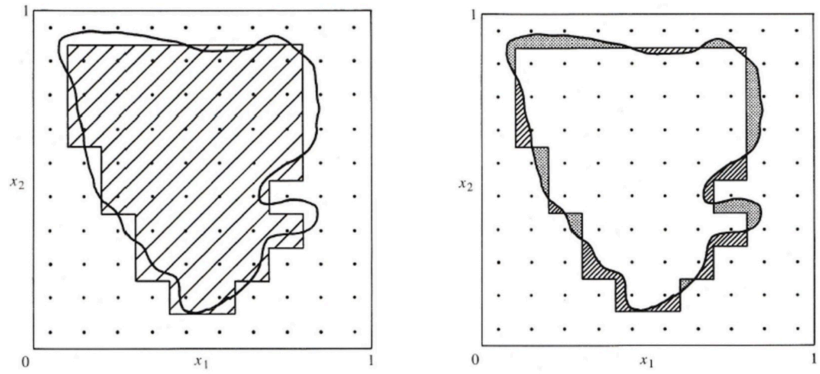


**Figure 4**

In the worst case, i.e. when errors do not cancel out, the error $|\lambda_{\text{grid}} - \lambda(R)|$ is bounded above by the total dark area (still from the picture on the right):

$$|\lambda_{\text{grid}} - \lambda(R)| \leq \text{dark area}$$

Let $s(R)$ be the length of the frontier of $R$, that is the set of points that delimit $R$. Loosely speaking, if $R$ is a volume in $m$ dimensions, it's frontier will typically be a $m-1$ dimensional object. Since here we are in dimension $m = 2$, the frontier is a 1D object, a line. We can use the "hyper-surface" of the frontier (the length of the line) to bound by above the dark area: the dark area cannot be longer than $s(R)$, nor thicker than $k$. In the worst case, the error on the estimate is:

$$\text{dark area} \leq s(R) \times k \tag{6}$$

Therefore we obtain a relation for the error on the estimate of the volume:

$$|\lambda_{\text{grid}} - \lambda(R)| \leq s(R) \times k = \frac{s(R)}{n^{\frac{1}{m}}} \tag{7}$$

Now we fix a precision requirement $\epsilon$ for the estimate, and we want to determine for what values of $n$ the requirement is met, since changing $n$ in this case means refining the grid. We have

$$|\lambda_{\text{grid}} - \lambda(R)| \leq \frac{s(R)}{n^{\frac{1}{m}}} \leq \epsilon \tag{8}$$

The precision requirement is thus surely met for those values of $n$ such that

$$n \geq n_{\text{grid}} \equiv \left(\frac{s(R)}{\epsilon}\right)^m$$

Here we see a big problem in this estimator of the volume based on points generated on a grid: the number of points it needs for a fixed precision $\epsilon$ grows exponentially with the dimension $m$ (provided the surface stays roughly the same). The problem becomes computationally expensive very quickly in high dimensionality.

### Computing Volumes By Using Monte Carlo Sampling

The aforementioned problem can be overcome using Monte Carlo sampling. That is, instead of using samples from a lattice, we sample $n$ times from a uniform distribution in $[0,1]^m$ (mind that we must sample each one of the $m$ components of the vector $\boldsymbol{x}^j = \left(x_1^j, ..., x_m^j\right)$, where $j = 1, ..., n$ is the sample index).

Recall that for Algorithm 1 to work, it is necessary to have an indicator function of $R$ that we called IsInside $\left(\mathbf{x}^j, R\right)$, which checks whether the sampled vector $\mathbf{x}^j$ lies within the region $R$ or not. Let's consider a single random sample $\boldsymbol{x}$. Here for convenience we change the name of this function to $\phi_R(\boldsymbol{x})$:

$$\phi_R(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \in R \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Since $\boldsymbol{x}$ is a uniformly distributed random variable in the box $[0,1]^m$, also $\phi_R(\boldsymbol{x})$ is a random variable. It is easy to check that its expected value is the volume $\lambda(R)$ itself. In fact, we have

$$\mathbb{E}\left[\phi_R(\boldsymbol{x})\right] \equiv \int_{[0,1]^m} d\boldsymbol{x}\ \phi_R(\boldsymbol{x}) = \int_R d\boldsymbol{x}\ 1 = \lambda(R).$$

The variance of $\phi_R(\boldsymbol{x})$ is also easy to compute:

$$\begin{aligned} \mathrm{Var}\left[\phi_R(\boldsymbol{x})\right] &\equiv \mathbb{E}\left[\left(\phi_R(\boldsymbol{x}) - \lambda(R)\right)^2\right] \\ &= \mathbb{E}\left[\phi_R(\boldsymbol{x})^2\right] - 2\mathbb{E}\left[\phi_R(\boldsymbol{x})\right]\lambda(R) + \lambda(R)^2 \\ &= \mathbb{E}\left[\phi_R(\boldsymbol{x})^2\right] - \lambda(R)^2 \\ &= \lambda(R)\left(1 - \lambda(R)\right) \end{aligned}$$

The last step uses the fact that $\phi_R(\mathbf{x})^2 = \phi_R(\mathbf{x})$ for all $\mathbf{x}$ since the only possible values are 0 or 1.

Notice that these are the mean and the variance of Bernoulli random variable with parameter $\lambda(R)$.

Now let's go back to our algorithm where we sample $n$ independent points $\boldsymbol{x}^j, j = 1, \ldots, n$. In order to characterize the random output from the algorithm, we have to study the random variable

$$S = \sum_{j=1}^{n} \phi_R\left(\boldsymbol{x}^j\right)$$

Using the linearity of expectations we have

$$\mathbb{E}[S] = \sum_{j=1}^{n} \mathbb{E}\left[\phi_R\left(\boldsymbol{x}^j\right)\right] = n\lambda(R)$$

Then, using the fact that the variance of the sum of independent variables is equal to the sum of the variances, we have

$$\mathrm{Var}[S] = n\lambda(R)(1-\lambda(R))$$

Actually, we can go further on and compute the full probability distribution of $S$, using the fact that $S$ is the sum of independent and identically distributed (i.i.d.) Bernoulli random variables. This leads to the binomial distribution:

$$P(S=s) = \binom{n}{s}\lambda(R)^s(1-\lambda(R))^{n-s}$$

Since the return value from the Monte Carlo algorithm is $\lambda_{\mathrm{MC}} = S/n$, we obtain the following result:

- $\lambda_{\mathrm{MC}}$ is an unbiased estimator of the true volume, meaning that its expected value is the true volume

$$\mathbb{E}[\lambda_{\mathrm{MC}}] = \frac{\mathbb{E}[S]}{n} = \lambda(R)$$

- has variance that shrinks with the number of samples $n$:

$$\mathrm{Var}[\lambda_{\mathrm{MC}}] = \frac{\lambda(R)(1-\lambda(R))}{n}$$

These two results give us the theoretical guarantee that increasing the sample size $\lambda_{\mathrm{MC}}$ gives us more and more accurate estimates of the true volume value $\lambda(R)$.

Now, in order to obtain an inequality that tells us the minimum $n$ needed to for $\lambda_{\mathrm{MC}}$ to satisfy a certain accuracy requirement, we will use the Chebyshev's inequality, which we state and prove in the following paragraph.

**Chebyshev's Inequality** Let $X$ be a scalar random variable with probability density function $f(x)$, expected value $\mathbb{E}[X] = 0$ and finite variance $\mathrm{Var}[X] = \mathbb{E}\left[X^2\right] = \sigma^2$. Then $\forall\beta > 0$ the Chebyshev's Inequality states

$$P\left(\frac{|X|}{\sigma} \geq \beta\right) \leq \frac{1}{\beta^2}$$

**Proof** Choose $\epsilon > 0$. Then:

$$\begin{aligned}
P(|X| \geq \epsilon) &= \int_{-\infty}^{-\epsilon} dx\, f(x) + \int_{+\epsilon}^{+\infty} f(x) \\
&\leq \int_{-\infty}^{-\epsilon} dx \frac{x^2}{\epsilon^2} f(x) + \int_{+\epsilon}^{+\infty} dx \frac{x^2}{\epsilon^2} f(x) && (x^2 \geq \epsilon \text{ in both integration domains}) \\
&\leq \int_{-\infty}^{+\infty} dx \frac{x^2}{\epsilon^2} f(x) && (\text{add the integral between } -\epsilon \text{ and } \epsilon, \text{ which is non-negative}) \\
&= \frac{\sigma^2}{\epsilon^2} && (\text{definition of } \sigma)
\end{aligned}$$

Set $\epsilon = \beta\sigma$. Then we have

$$P(|X| \geq \beta\sigma) \leq \frac{\sigma^2}{\beta^2\sigma^2} = \frac{1}{\beta^2}$$

that means

$$P\left(\frac{|X|}{\sigma} \geq \beta\right) \leq \frac{\sigma^2}{\beta^2\sigma^2} = \frac{1}{\beta^2} \qquad \blacksquare$$

Now choose the $X$ variable in the Chebyshev's inequality to be $X = \lambda_{MC} - \lambda(R)$. Then from the Chebyshev's inequality we have that:

$$P(|\lambda_{MC} - \lambda(R)| \leq \epsilon) \geq 1 - \frac{1}{\epsilon^2} \frac{\lambda(R)(1 - \lambda(R))}{n} \equiv 1 - \delta$$

where in the last equation we defined the probability confidence level

$$\delta \equiv \frac{1}{\epsilon^2} \frac{\lambda(R)(1 - \lambda(R))}{n}$$

If both $\epsilon$ and $\delta$ are small, the interpretation is that we can be almost sure ("with confidence $1 - \delta$") that our estimate will be a good one, with and error of at most $\epsilon$.

For any desired pair of values $\epsilon$ and $\delta$, we can then find a value of $n$ that ensures that the above inequality holds. Indeed, we find that our sample size $n$ has to satisfy:

$$n \geq \frac{\lambda(R)(1 - \lambda(R))}{\delta \epsilon^2}$$

Since $0 \leq \lambda(R) \leq 1$ then we have $\lambda(R)(1 - \lambda(R)) \leq \frac{1}{4}$, and finally we find that the requirement for $n$ is given by

$$n \geq n_{\text{MC}} = \frac{1}{4\delta\epsilon^2}$$

Note that this bound does not depend on $m$. This is the crucial result. In contrast, recall that the one for the grid algorithm was

$$n_{\text{grid}} = \left(\frac{s(R)}{\epsilon}\right)^m$$

The Monte Carlo sampling algorithm takes polynomial time in $m$, since all its iterations are polynomial in $m$ and the number $n$ of iterations does not depend on $m$. On the contrary, the grid algorithm, which samples $R$ with a lattice, takes exponential time. The advantage of the Monte Carlo method is more and more relevant when $m$ is large. This happens more often than one might naively expect: for example, a typical occurrence is when $m$ represents the number of variables of a combinatorial problem.

One drawback is that of course, since MC is a randomized algorithm, we always have a slight chance $\delta$ to be unlucky, i.e. that our estimate will be wrong – farther than $\epsilon$ from the truth. But as we've seen, with enough samples this becomes negligible.

## 1.2 Counting objects

Another typical application of the Monte Carlo method is for counting objects with specific properties in a (very large) discrete set. Consider the following example.

Consider a vertex of the $m$-dimensional hypercube $\boldsymbol{x} = (x_1, x_2, ..., x_m)$, where $x_i \in \{0, 1\}\ \forall i = 1, ..., m$. Let's call $\mathcal{X}$ the set of all points $\boldsymbol{x}$. Consider then a set of $k$ inequalities (constraints) of the form

$$a_{j_1} x_1 + a_{j_2} x_2 + ... + a_{j_m} x_m \leq b_j \qquad \forall j = 1, .., k$$

We can be interested in counting how many points in $\mathcal{X}$ satisfy all the inequalities (such points form the feasible region of an *integer programming* problem). Alternatively, we may be interested in counting how many points in $\mathcal{X}$ satisfy at least one of the inequalities (*disjunctive programming*) or a prescribed number of inequalities. If a point satisfies such constraints we say that it has the property $P$.

The number of points in $\mathcal{X}$ is $|\mathcal{X}| = 2^m$ and an explicit enumeration of all of them is out of question even for relatively small values of $m$. This is the reason for estimating the number of points with the desired property, instead of explicitly counting them. For this purpose we can resort to the Monte Carlo method. We need two procedures:

- a procedure to generate elements from the ground set $\mathcal{X}$ with uniform probability $1/|\mathcal{X}|$;
- a procedure to test whether an element $x \in \mathcal{X}$ has the property $P$ or not.

Therefore an immediate modification of algorithm 1 can be formulated as follows:

**Algorithm 2** Monte Carlo counting

**Require:** $\mathcal{X}, P, n$
  $S \leftarrow 0$
  **for** $j \leftarrow 1, ..., n$ **do**
    $\mathbf{x} \leftarrow$ GeneratePoint(X) ▷ uniformly pick from the ground set $X$
    **if** HasProperty($\mathbf{x}, P$) **then**
      $S \leftarrow S + 1$
    **end if**
  **end for**
  **return** $S/n$

## 1.3 Multidimensional integration

Multi-dimensional integrals can be interpreted as expectations of a function of a vector-valued random variable and uniformly distributed (in the integration volume) random variable $\boldsymbol{X}$:

$$\int_{a_1}^{b_1} dx_1 ... \int_{a_m}^{b_m} dx_m \; f(x_1, ..., x_m) = V \cdot \mathbb{E}[f(\boldsymbol{X})]$$

where $V$ is the integration volume $(b_1 - a_1) \times (b_2 - a_2) \times \cdots \times (b_m - a_m)$. Given $n$ samples $\left\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n))}\right\}$ uniformly distributed in $V$, we can approximate the expectation with

$$\mathbb{E}[f(x)] \simeq \frac{1}{n} \sum_{j=1}^{n} f\left(\mathbf{x}^{(j)}\right)$$

This is the Monte Carlo way of estimating integrals. In the particular case of $f$ being an indicator function we recover the volume computing method discussed previously.

Note that the Monte Carlo estimation could be made more efficient concentrating the sampling (which in such case won't be uniform anymore) in the regions dominating the integral (larger values of $f$). This is called *importance sampling*.

# 2 Markov Chain Monte Carlo

In this Section we deal with stochastic dynamical systems: variables that evolve in time and whose evolution is not (only) driven by some deterministic law since there is also some randomness at play.

## 2.1 Stochastic Processes

Consider a possibly infinite sequence of random variables $\left(X^0, X^1, X^2, \dots, X^t, \dots\right)$ indexed by an index $t = 0, 1, \dots$ that we commonly refer to as *time,* so that we interpret the sequence as the (stochastic) change of $X$ over time. The random variables take value in some state space $\mathcal{X}$, i.e. $X^t \in \mathcal{X}$. For the time being we don't make any assumptions on $\mathcal{X}$: it can either be some finite set of elements, or an infinite but discrete space like the set of natural numbers $\mathbb{N}$, or an infinite and continuous space like $\mathbb{R}$, or some $d$-dimensional space like the Euclidean space $\mathbb{R}^d$ (in the latter case, $X^t$ will be a random vector and we may denote it with $\boldsymbol{X}^t$).

Such sequences are know as stochastic processes, and they are ubiquitous in nature for two main reasons: 1) because the laws of physics at the microscopic level (involving elementary particles) are non-deterministic; 2) because at the microscopic level many phenomena involve a large number of actors interacting in such complicated ways that for any practical purpose they can be modeled as stochastic process.

An example of the latter case is given by finance. The price of a final asset, let's say Google's stocks, evolves in time as a result of the sells of Google's products but also has the result of an large number of interactions among traders, rumors, irrational expectations... in such a complex way that no one is able to predict the future evolution of the prices. Looking at the daily or hourly fluctuations of the prices one can't help noticing that, even though the system is purely deterministic it behaves as affected by stochasticity.

Unfortunately in nature, since we cannot go back in time, we often observe a unique realization of the stochastic process, that is a just a unique value for $X^0$, another for $X^1$, and so on. In other words we observe only a single *trajectory* out of the many possible ones, each of which can be more or less likely. In *computer simulations*, instead, one can sample again and again the same processes, obtaining different trajectories and gaining valuable information on the structure of the process itself.

Stochastic processes are entirely defined by the probability of observing each trajectory. For a given trajectory $\left(x^0, x^1, x^2, \dots\right)$ the corresponding probability is written as

$$P\left(X^0 = x^0, X^1 = x^1, X^2 = x^2, \dots\right) \equiv P\left(x^0, x^1, x^2, \dots\right). \tag{10}$$

When there is no ambiguity, we will typically use the more concise notation on the r.h.s. instead of the more formally correct notation on the l.h.s.. In the case of continuous random variables, the expression above has to be intended as a probability density.

From the probability of a whole trajectory, that is the joint probability over the variables $\left(X^0, X^1, \dots\right)$, one can derive the probabilities over a subset of the variables by marginalization, i.e. by taking sums over all the possible values of all the variables except the ones we're interested in. For example, the probability that the process at time $t$ takes some value $x^t$, i.e. $P\left(X^t = x^t\right)$ is obtained (assuming discrete states) by summing over the all possible values at all times except for $t$:

$$P\left(x^t\right) = \sum_{x^0} \cdots \sum_{x^{t-1}} \sum_{x^{t+1}} \sum_{x^{t+2}} \dots P\left(x^0, x^1, \dots\right) \tag{11}$$

$$= \sum_{x^0, \dots, x^{t-1}, x^{t+1} \dots} P\left(x^0, x^1, \dots\right) \tag{12}$$

This amounts to sum over all possible trajectories that at time $t$ take values $x^t$, each trajectory weighted by the corresponding probability.

It is often the case that it is hard to explicitly give the probability distribution $P\left(x^0, x^1, x^2, \dots\right)$ for the process, while it may be very easy to define the process by stating how to generate the random variable $X^t$ given the realization of $X^{t-1}, X^{t-2}, \dots$ in the previous steps. This stochastic rule for the evolution implicitly defines a distribution $P\left(x^0, x^1, x^2, \dots\right)$, which we may or may not be able to compute.

Let's now give some examples of stochastic process, starting from a trivial one with no time correlation and arriving to a more complicated one with long term memory.

**Coin Tosses**

In this simple process, we toss a fair coin at each step. The result of the toss is either head or tail, therefore we can define the state space with $\mathcal{X} = \{H, T\}$ (or equivalently $\mathcal{X} = \{0, 1\}$). Therefore, possible trajectories are of the form $(H, H, T, T, H, H, H, H, H, T, H, T, T, \dots)$. It is easy to see the since the coin is fair and there is no history dependence, all trajectories are equiprobable. In fact, if we consider the process up to some time $t$, corresponding to $t+1$ tosses, the probability of any trajectory $\left(x^0, \dots, x^t\right)$ is easy to compute since each toss is independent. Therefore, the joint probability factorizes in the probabilities for the single draws and we have

$$P\left(x^0, \dots, x^t\right) = \prod_{t'=0}^{t} P\left(x^{t'}\right) = \frac{1}{2^{t+1}} \tag{13}$$

**Random Walk (RW)**

A random walk in $\mathbb{Z}^d$ is a walk that starts from the origin of the $d$-dimensional lattice, and takes a step of length one in a random direction, choosing uniformly at random among $2d$ the possible the available directions. This is an example of a possible trajectory for a walk in $\mathbb{Z}^2$: $\boldsymbol{x}^0 = (0,0)\,,\ \boldsymbol{x}^1 = (-1,0)\,,\ \boldsymbol{x}^2 = (-1,1)\,,\ \boldsymbol{x}^3 = (-1,0)\,,\ \boldsymbol{x}^4 = (-2,0)\,, \dots$.

The process can be written as follows

$$\boldsymbol{X}^0 = \boldsymbol{0}, \tag{14}$$

$$\boldsymbol{X}^{t+1} = \boldsymbol{X}^t + \boldsymbol{D}^t \qquad \text{for } t \geq 0, \tag{15}$$

where $\boldsymbol{D}^t$ is a random vector chosen uniformly at time $t$ from the set

$\{(1,0,0,\dots)\,, (-1,0,0,\dots)\,, (0,1,0,\dots)\,, (0,-1,0,\dots)\,, \dots\}$ containing $2d$ elements. Notice that for a random walk, the probability of visiting some state at time $t+1$ given the states visited up to time $t$, depends only on the state at time $t$ and not on the whole past trajectory:

$$P\left(x^{t+1} \mid x^t, x^{t-1}, \dots, x^0\right) = P\left(x^{t+1} \mid x^t\right) \tag{16}$$

**Self-Avoiding Walk (SAW)**

A SAW is a random walk that cannot visit twice the same site. Therefore, at each step it visits uniformly at random one of the neighboring sites that it hasn't already visited (they can be less then $2d$). If at a certain step there are no more directions available then the process stops. This is an example of a process where the probability of going on a certain state at time $t+1$ depends on the *whole* past trajectory.

## 2.2 Markov Chains

Generally, stochastic processes can be very complicated and difficult to analyze. Here we consider a restricted class of stochastic processes, named Markov Chain (MC) processes, that are much easier to analyze while still being rich enough to model many real world situations. Also, as we will see in the following paragraphs, one can carefully design and simulate Markov Chains in order to perform hard computational tasks such as sampling from very complicated distributions or solving hard optimization problems.

Markov Chains are defined by the property of being *memory-less* stochastic processes, that is stochastic processes in which the probability of the next state given the history of the process depends *only on the current state*. This is called the Markov Property, and is formally written as

$$P\left(x^{t+1} \mid x^t, x^{t-1}, ..., x^0\right) = P\left(x^{t+1} \mid x^t\right) \tag{17}$$

Being $P\left(x^{t+1} \mid x^t\right)$ a conditional probability, we note that it is also normalized, meaning that the probability of transition from a state $x^t$ to any possible state ($x^t$ included) is 1, that is $\sum_{x^{t+1}} P\left(x^{t+1}|x^t\right) = 1$.

The coin toss example from last paragraph falls trivially within the Markov Chain (MC) family, since each new toss does not depend at all on the results from the last tosses, not even the last one. Random walks instead are more dignified example of a MC, with dependence of the next state on the current state (and no other past states). This conditional dependence can be easily read in equation (15).

On the other hand, self-avoiding walks are not Markov chains: equation (17) does not hold, since the knowledge of each of the past states $x^0, x^1, \ldots$ is relevant to determine which states can be visited at time $t+1$ and which cannot.

Thanks to the Markov property, the probability of the a trajectory takes a simple structure. Consider a Markov chain $\left(X^0, X^1, \ldots, X^t\right)$ running up to some time $t$. We have

$$P\left(x^0, x^1, \ldots, x^t\right) = P\left(x^0\right) \prod_{t'=0}^{t-1} P\left(x^{t'+1} \mid x^{t'}\right). \tag{18}$$

Therefore, everything is encoded in the initial state distribution $P\left(x^0\right)$ and in the *transition probabilities* $P\left(x^{t'+1}|x^{t'}\right)$. From the sampling point of view, this is very convenient: in order to sample a Markov chain trajectory, one first samples $x^0$ according to $P\left(x^0\right)$, then extracts $x^1$ according to $P\left(x^1|x^0\right)$, $x^2$ from $P\left(x^2|x^1\right)$ and so on.

Moreover, we have the following iterative rule that links the marginal distribution probability $P\left(x^t\right)$ at time $t$ – i.e. the probability distribution of the random variable $X^t$ - to the one at time $t+1$:

$$P\left(x^{t+1}\right) = \sum_{x^t} P\left(x^{t+1}|x^t\right) P\left(x^t\right) \qquad \forall x^{t+1} \in \mathcal{X} \tag{19}$$

Those last equations simply say the all the "probability mass" in the state $x^{t+1}$ at time $t+1$ is just given by the mass transitioning to $x^{t+1}$ from any possible state at previous time.

### Finite State Spaces

When the state space $\mathcal{X}$ is finite (we call this case *finite* Markov chain) it may be convenient to reframe the concept of Markov processes in the framework of linear algebra. Let $M$ be the size of $\mathcal{X}$, i.e. the total number of states, $M = |\mathcal{X}|$. We can label these states with numbers going from 1 to $M$. Therefore, for each element $x \in \mathcal{X}$ there is a corresponding label $i \in \{1, \ldots, M\}$ and we can essentially use the either $i$ or $x$ to denote the same state.

The transition probabilities at time $t$, $P\left(x^{t+1}|x^t\right)$ can then be encoded into $M \times M$ matrix $P_{ij}^{(t)}$ that we call *transition matrix,* whose elements are given by

$$P_{ij}^{(t)} = P\left(X^{t+1} = i \mid X^t = j\right) \qquad \forall i, j. \tag{20}$$

Notice that this represents the probability of a transition from $j$ to $i$, so the indices are in the order destination-source. This may seem confusing but it is convenient, as we shall see.

Since $P^{(t)}$ represents conditional probabilities, it has the property of being column-normalized, in the sense that $\sum_i P_{ij}^{(t)} = 1$ for each column $j$. We say that $P^{(t)}$ is a "stochastic matrix" because of this property.

On the other hand, the marginal probability distribution $P\left(x^t\right)$, can be encoded into a vector of size $M$, that we call $\boldsymbol{p}^t$ defined as follows by its components $p_i^t$:

$$p_i^t = P\left(X^t = i\right) \qquad \forall i. \tag{21}$$

Up to this point, it seems like we have introduced some redundant definitions without gaining anything. A first understanding of the convenience of this matrix-vector formalism derives from rewriting Eq. (19) in the new formalism:

$$p_i^{t+1} = \sum_{j=1}^{M} P_{ij}^{(t)} p_j^t \qquad \forall i. \tag{22}$$

This is nothing else than a matrix-vector product, that one can conveniently express in linear algebra notation:

$$\boldsymbol{p}^{t+1} = P^{(t)}\boldsymbol{p}^t$$

Using multiple times the last equation, we can also write the distribution at time $t$ as function of the one at time $t = 0$:

$$\boldsymbol{p}^t = P^{(t-1)}P^{(t-2)}\dots P^{(0)}\boldsymbol{p}^0 \tag{23}$$

The matrices can be applied to the vector in order, producing a new vector every time; however, because of associativity, we can also instead simply multiply all the transition matrices together and obtain directly a matrix with the transition probabilities from the initial state to the state at time $t$.

From now on we will use the matrix-vector notation and the probability distribution notation indistinctly when discussing finite-space Markov chains.

### Stationary Markov Chains

If the transition probabilities do not depend on time indexes, the Markov chain is said to be *time-independent* or *time-homogeneous* or *stationary.* The stationary condition can be written as

$$P\left(X^{t+1} = x'|X^t = x\right) = P\left(X^{t'+1} = x'|X^{t'} = x\right) \quad \forall\ x, x' \in \mathcal{X} \text{ and } \forall t, t'. \tag{24}$$

For finite state spaces, this also means that we can encode the process into a single transition matrix $P_{ij}$, so that we have

$$\boldsymbol{p}^{t+1} = P\,\boldsymbol{p}^{t} \tag{25}$$
$$= P^{t}\,\boldsymbol{p}^{0}. \tag{26}$$

Notice that $P^t$ in last is the product of $t$ matrices $P$, while the $t$ in $\boldsymbol{p}^t$ is just a time index. Throughout this notes we will mostly consider stationary Markov processes, the only exception being Simulated Annealing that is non-stationary due to the time dependence of the temperature.

### 2.2.1 Stationary Distributions

The concept of stationary distribution is central to the study and the applications of Markov chains. Given a stationary Markov chain identified by the transition probabilities $P(x' \mid x)$, a probability distribution $\pi(x)$ is said to be a *stationary distribution* for the process if it reproduces itself during the temporal dynamics (19) (or equivalently Eq. (25)); namely, $\pi(x)$ must satisfy the following condition:

$$\pi(x') = \sum_{x \in \mathcal{X}} P(x' \mid x)\,\pi(x) \qquad \forall x' \in \mathcal{X},$$

which in vectorial form reads

$$\boldsymbol{\pi} = P\boldsymbol{\pi}.$$

Notice that last equation tells us that if the stationary distribution $\boldsymbol{\pi}$ exists, than it is an eigenvector of the transition matrix $P$ with eigenvalue 1.

At this point, a few questions concerning stationary distributions naturally arise: 1) does any Markov process $P$ admits a stationary distribution? 2) is such distribution unique? 3) starting from the an initial condition $P(x^0)$, does $P(x^t)$ converge on $\pi(x)$ at large times? The answer to all the 3 questions is yes, once we restrict the family of process we consider to *ergodic* processes. A precise definition of ergodicity is technically involved, therefore here we just convey the general intuition; the interested reader can find more details in any specialized stochastic processes textbook.

Essentially, the idea of ergodicity is that the stochastic dynamics is not trapped in some regions of the state space for indefinite time, but is free to explore the whole space as time goes by. Moreover, in an ergodic process there is no periodicity. Formally we state that a finite Markov Chain is *ergodic* if it has the following properties:

- **Irreducibility**: the chain can go from any state $j$ to any state $i$ in a finite number of steps with probability larger than 0. This means that for each pairs of states $i$ and $j$ there exists an integer $k > 0$ such that $\left(P^k\right)_{ij} > 0$.
- **Aperiodicity**: there is no state that is visited by the stochastic process with finite probability only periodically, that is at multiples of a fixed period $k > 1$.

If the Markov chain $P$ is ergodic, a theorem due to Perron and Frobenius implies that a stationary distribution exists, and it is unique. Calling $\boldsymbol{\pi}$ such stationary distribution (in vector notation), we also have that, regardless of the initial state distribution $\boldsymbol{p}^0$, the state distribution $\boldsymbol{p}^t$ converges to $\boldsymbol{\pi}$ at large times, that is:

$$\boldsymbol{p}^t = P^t \boldsymbol{p}^0 \xrightarrow[t \to +\infty]{} \boldsymbol{\pi}$$

One can show that the convergence is exponentially fast, and that the convergence rate is given by the second largest (in modulus) eigenvalue of $P$ (the first eigenvalue being 1).

**Averages in ergodic systems**

A very important property of ergodic processes is that *time averages* can be replaced by *ensemble averages* over the stationary distribution (actually, this is the very definition of an ergodic process). This means, that for any *observable* $f(x)$ (a function on $\mathcal{X}$) and for any trajectory $(x^t)_{t\geq 0}$ sampled according to an ergodic process with stationary distribution $\pi$, we have (almost surely in probability) that

$$\begin{aligned}\bar{f} &\equiv \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} f\left(x^t\right) \\ &= \mathbb{E}_{X\sim\pi} f(X) \equiv \sum_{x\in\mathcal{X}} \pi(x) f(x)\end{aligned}$$

## 2.3 Global Balance and Detailed Balance

Consider now a stationary distribution $\boldsymbol{\pi}$ for the Markov chain with transition matrix $P$, that is a distribution (in vector form, since we consider finite space states now) satisfying the equation

$$\pi_i = \sum_j P_{ij}\pi_j \qquad \forall i.$$

Now we plug in the l.h.s. the identity in the form $1 = \sum_j P_{ji}$ which stems from the fact that $P$ is a stochastic matrix representing a conditional probability: summing over a column of the matrix $P_{ji}$ is the probability of going *anywhere* from the state $i$, which must be 1. If we then subtract from both sides a term $P_{ii}\pi_i$ we arrive to the following equation, known as *global balance* condition:

$$\sum_{j\neq i} P_{ji}\pi_i = \sum_{j\neq i} P_{ij}\pi_j \qquad \forall i. \tag{27}$$

This set of equations (we have one for each $i$) states than when stationarity is reached, whatever "flows out" of a given state $i$ to all other states is perfectly balanced by what "flows in" from all other states into $i$. The global balance condition is a necessary and sufficient condition for stationarity.

Now we give a stronger condition called *detailed balance* condition that, being stronger, doesn't apply to any stationary distribution and therefore is only a sufficient condition for stationarity. The *detailed balance* condition reads

$$P_{ji}\pi_i = P_{ij}\pi_j \qquad \forall i,j.$$

This form of balance is "detailed" because the balance of out-flow and in-flow applies to each pair of states $i$ and $j$. Summing both sides of last equation over $j$ it is easy to see that detailed balance implies global balance, which in turn implies stationarity. Checking if the detailed balance condition holds for a certain distribution $\boldsymbol{\pi}$ and chain $P$ is typically easier than checking the global balance condition, therefore this condition is typically actually used in practical applications, as we will see in the following paragraph.

## 2.4 Application I: Spanning Trees Sampler

For a given graph $G = (V, E)$, a spanning tree $T$ is a subgraph of $G$ containing all nodes of $G$ (this the spanning part) and no cycles (since it is a tree). Generally, a graph with $N = |V|$ nodes may contain an exponentially large

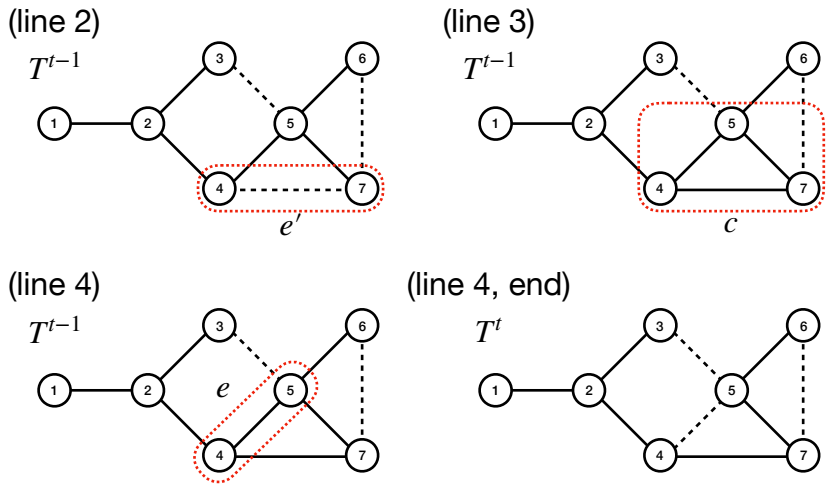


**Figure 5:** Steps 2-4 of the spanning tree sampler algorithm.

(in $N$) number of spanning trees. Let's denote with $\mathcal{T}$ the set of spanning trees, $T \in \mathcal{T}$. In this section we develop a Monte Carlo algorithm for sampling uniformly at random spanning trees from $\mathcal{T}$. This is not a trivial task, since we don't want to enumerate in advance all the possible spanning trees. First we present the algorithm, then we will check a-posteriori that the algorithm (approximately) samples from the uniform distribution by showing that the uniform distribution satisfies detailed balance for the Markov chain induced by the sampling procedure in the algorithm.

The algorithm goes as follows:

1. Start at time $t = 0$ with some initial spanning tree $T^0$.
2. At time step $t$ select uniformly at random an edge $e'$ that is in the graph but not in $T^{t-1}$.
3. Add the edge $e'$ to $T^{t-1}$. The addition will form a cycle $c$ in the tree.
4. Remove an edge $e$ chosen uniformly at random in the cycle $c$. This operation will produce again a tree subgraph that we call $T^t$.
5. Increase the time counter: $t = t + 1$.
6. Repeat steps (2-5) up to time $t_{\rm MAX}$.

Steps 2-4 are represented in Fig. 5. Notice that in the step 4 it could be the case that $e = e'$: for this event $T^t = T^{t-1}$. This stochastic algorithm is presented as pseudocode in Alg. 3.

Is the result from this stochastic algorithm a *uniformly distributed* spanning tree? That is, is it true that $P\left(T^{t_{\rm MAX}} = T\right) = \frac{1}{|\mathcal{T}|} \quad \forall\, T \in \mathcal{T}$ ?

- If $t_{\rm MAX}$ is small this is not the case, since $T^{t_{\rm MAX}}$ is highly correlated to $T^0$.
- If $t_{\rm MAX} \gg 1$, namely in the large time limit, $T^{t_{\rm MAX}}$ is uniformly distributed, as we will now show.

**Algorithm 3** Spanning Tree Generator

**Require:** a graph $G = (V, E)$, an initial spanning tree $T^0$ for $G$, a stopping time $t_{\text{MAX}}$
**Ensure:** an (approximately) uniformly distributed tree $T^{t_{\text{MAX}}}$
  **for** $t \leftarrow 1, ..., t_{\text{MAX}}$ **do**
    $e' \leftarrow \text{SelectRandEdge}(E \setminus T^{t-1})$
    $c \leftarrow \text{FindCircuit}(T^{t-1} \cup \{e'\})$
    $e \leftarrow \text{SelectRandEdge}(c)$
    $T^t \leftarrow T^{t-1} \cup \{e'\} \setminus \{e\}$
  **end for**
  **return** $T^{t_{\text{MAX}}}$

The described Markov chain is ergodic (finite, irreducible, aperiodic), hence if we find a distribution that satisfies the detailed balance condition, that distribution is the stationary distribution for the Markov chain. Let $\pi(T) = \frac{1}{|\mathcal{T}|}$ be the uniform distribution and let's check that it is indeed the stationary distribution of the process; we will do so by proving that the detailed balance condition holds.

Let's call $P(T' \mid T)$ the transition probability of Markov chain for going from tree $T$ to tree $T'$. The detailed balance condition reads:

$$P(T' \mid T)\,\pi(T) = P(T \mid T')\,\pi(T')$$
$$P(T' \mid T)\,\frac{1}{|\mathcal{T}|} = P(T \mid T')\,\frac{1}{|\mathcal{T}|}$$
$$P(T' \mid T) = P(T \mid T')$$

Therefore we just have to check that the transition probability is symmetric. This is trivial if $T' = T$. If instead there is an edge $e'$ in $T'$ but not in $T$ and an edge $e$ in $T$ but not in $T'$, we have that:

- $P(T' \mid T) = P(\text{choose } e') \times P(\text{eliminate } e) = \frac{1}{|E|-|T|} \cdot \frac{1}{|c|}$
- $P(T \mid T') = P(\text{choose } e) \times P(\text{eliminate } e') = \frac{1}{|E|-|T'|} \cdot \frac{1}{|c|}$

where $|E|$ is the number of edges in the graph $G$, $|T|$ is the number of edges in the spanning tree and $|c|$ is the number of edges in the cycle $c$ defined in line 3. Since $|T'| = |T|$ the transition probabilities are equal and therefore detailed balance holds. The uniform distribution is the stationary one and starting from any $T^0$ we finally have

$$p^t(T) \xrightarrow[t \to +\infty]{} \pi(T) = \frac{1}{|\mathcal{T}|} \qquad \forall T.$$

## 2.5 Application II: Google's PageRank

The Google's PageRank algorithm was invented by L. Page and S. Brin in 1996 to rank the popularity of web pages and was then implemented as the base of their famous search engine. The PageRank algorithm is based on the model of a user navigating the web going from hyperlink to hyperlink at random.

Consider the web as a directed graph $G = (V, E)$ where the nodes are the web pages and the directed edges are the hyperlinks from one page to the other. Let $A$ be the adjacency matrix of the graph:

$$A_{ij} = \begin{cases} 1 & \text{if } (j \to i) \in E \\ 0 & \text{otherwise} \end{cases}$$

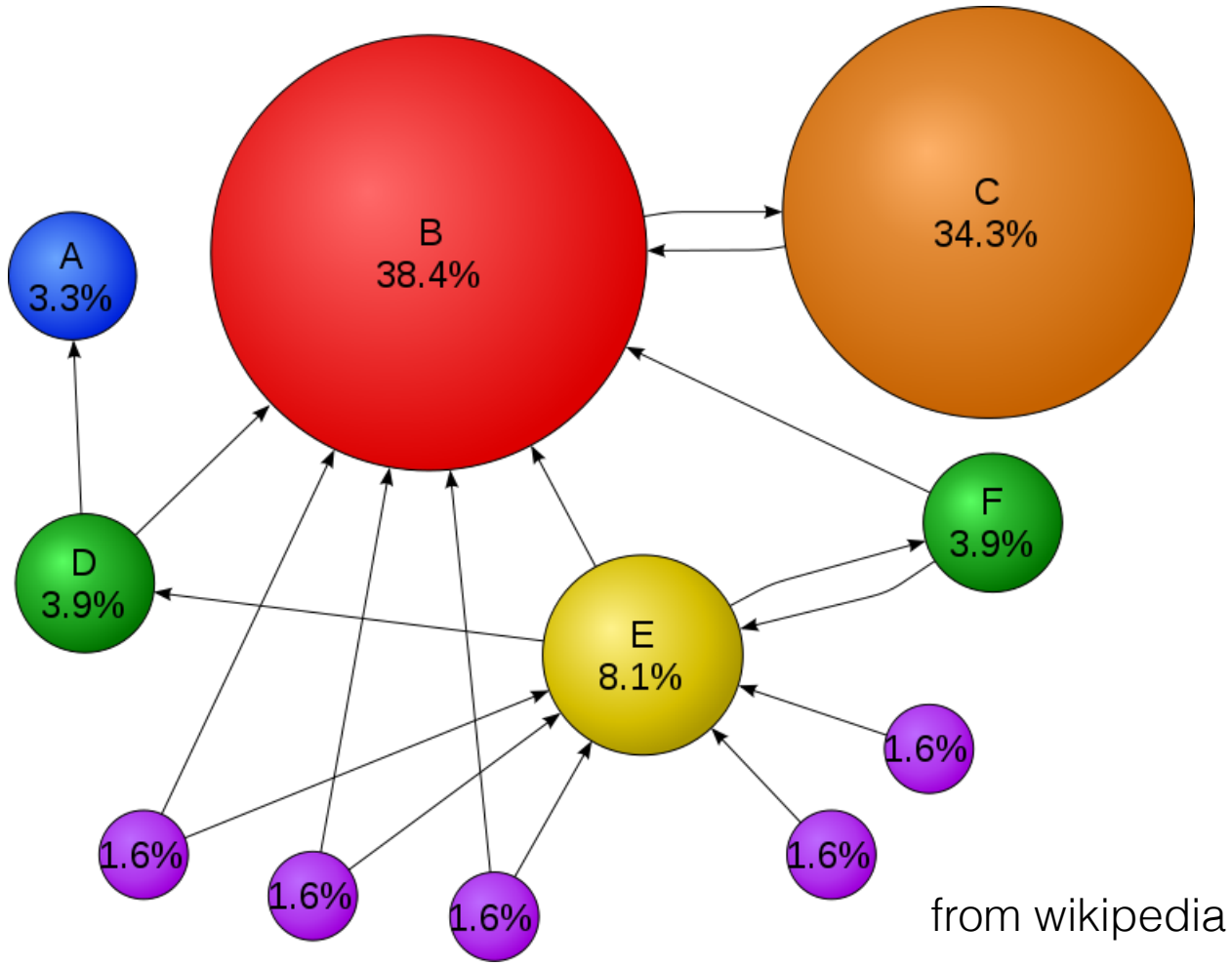


**Figure 6:** PageRanks for a simple network, expressed as percentages (Google uses a logarithmic scale). Page $C$ has a higher PageRank than Page $E$, even though there are fewer links to $C$; the one link to $C$ comes from an important page and hence is of high value. If web surfers who start on a random page have an 85% likelihood of choosing a random link from the page they are currently visiting, and a 15% likelihood of jumping to a page chosen at random from the entire web, they will reach Page $E$ 8.1% of the time. (The 15% likelihood of jumping to an arbitrary page corresponds to a damping factor of 85%.) Without damping, all web surfers would eventually end up on Pages $A$, $B$, or $C$, and all other pages would have PageRank zero. In the presence of damping, Page $A$ effectively links to all pages in the web, even though it has no outgoing links of its own.

Let $k_j$ be the outgoing degree of vertex $j$:

$$k_j = \sum_i A_{ij}.$$

A "random web surfer" jumps from page to page choosing among the links with equal probability. Hence, this is a Markov chain with transition matrix

$$M_{ij} = \frac{A_{ij}}{k_j}$$

The described Markov chain is ergodic (finite, irreducible, aperiodic) and hence there exists a probability distribution $\boldsymbol{\pi}$ such that:

$$\boldsymbol{\pi} = M\boldsymbol{\pi}$$

At a certain time $t$, the probability of visiting a certain page $i$, will be given by $p_i^t = \left(M^t \boldsymbol{p}^0\right)_i$. At large times, this probability will be closer and closer to the stationary distribution. Therefore the PageRank algorithms defines the page-rank (a measure of popularity) of a webpage $i$ as the value $\pi_i$. For a more realistic measure of the popularity of $i$, it is possible to introduce a damping coefficient $d$ ($0 \leq d \leq 1$), which represents the probability that a bored surfer jumps at random to any page in the web instead of going through a hyperlink in the current page. Let $N = |V|$ be the total number of nodes in the graph. With the addition of $d$, the transition matrix becomes

$$M_{ij}^{(d)} = \frac{1-d}{N} + d\frac{A_{ij}}{K_j}$$

PageRank scores for a small network of web-pages are represented in Fig. 6.

## 2.6 Metropolis Algorithm

**Algorithm 4** Metropolis Algorithm

**Require:** Initial configuration $x^0$, time $t_{max}$
**Ensure:** A configuration $x$ approximately distributed as $\pi$

$x \leftarrow x^0$
**for** $t = 1, ..., t_{max}$ **do**
  $x' \leftarrow$ sample from $g(\cdot|x)$ ▷ *propose a move from $x$ to $x'$*
  $a \leftarrow \min\left(1, \frac{\pi(x')}{\pi(x)} \frac{g(x|x')}{g(x'|x)}\right)$
  $r \leftarrow$ sample from Uniform$(0, 1)$
  **if** $r \leq a$ **then** ▷ *accept the move*
    $x \leftarrow x'$
  **end if**
**end for**
**return** x

We now turn the problem around: instead of asking ourselves what is the stationary distribution of a Markov chain, we consider a target distribution $\pi(x)$ and our objective is to devise a Markov chain that has $\pi(x)$ as its stationary distribution. In other words, we want to find a Markov chain that allows us to sample from the given distribution $\pi(x)$. This problem is very common in high-dimensional statistics where we know the form of $\pi(x)$ (typically the posterior distribution of the parameters of some model given some observed data) but sampling

from it is not easy. The problem of sampling from some $\pi(x)$ arises also in the simulated annealing optimization algorithms, as we will see later in these notes.

In order to do devise a Markov chain that at long times samples from $\pi(x)$, we let the detailed balance condition hold:

$$P(x' \mid x)\,\pi(x) = P(x \mid x')\,\pi(x') \tag{28}$$

$$\Rightarrow \frac{P(x' \mid x)}{P(x \mid x')} = \frac{\pi(x')}{\pi(x)} \tag{29}$$

We then split the transition probability $P(x' \mid x)$ in two parts

$$P(x' \mid x) = g(x' \mid x)\,A(x', x) \tag{30}$$

where:

- $g(x' \mid x)$ is called the *proposal kernel.* This is a properly normalized conditional probability distribution, that is $\sum_{x'} g(x' \mid x) = 1$.
- $A(x', x)$ is called the *acceptance ratio.* For each choice of $x'$ and $x$, this is a number $0 \leq A(x', x) \leq 1$.

It turns out that the decomposition (30) can be interpreted as follows: the probability of transitioning from a certain state $x$ to a different state $x'$ is equal to the probability of sampling $x'$ (the proposal) from the distribution $g(\bullet \mid x)$, times the probability of accepting the proposal, this given by $A(x', x)$.

Once we have this $g$ and $A$ decomposition, we have to link it to the target distribution $\pi$ in order to solve our task. The link is done by the detailed balance condition. We keep $g(x' \mid x)$ generic, since we want the proposal kernel to be flexible, and obtain the following equation from the detailed balance condition for

$$\frac{g(x' \mid x)\,A(x', x)}{g(x \mid x')\,A(x, x')} = \frac{\pi(x')}{\pi(x)} \tag{31}$$

$$\Rightarrow \frac{A(x', x)}{A(x, x')} = \frac{g(x \mid x')\,\pi(x')}{g(x' \mid x)\,\pi(x)} \qquad \forall x \neq x'. \tag{32}$$

There are many different choices for the function $A(x', x)$ that satisfy last equation. The Metropolis choice of $A(x', x)$ reads

$$A(x', x) = \min\left(1, \frac{g(x \mid x')\,\pi(x')}{g(x' \mid x)\,\pi(x)}\right) \tag{33}$$

It can be shown that this rule satisfies the detailed balance condition (check prof. Baldassi notes). For any target distribution $\pi(x)$ and any proposal kernel $g(x' \mid x)$, using Eq. (33) and Eq. (30) we can construct a Markov chain that has $\pi(x)$ as its asymptotic distribution. The problem of sampling from $\pi$ is solved by starting from any state and running the Markov chain for long time. The sampling procedure for this Markov chain is called Metropolis algorithm and presented in Alg. 4.

**Consideration 1** Typically $g$ is chosen to be symmetric, that is $g(x' \mid x) = g(x \mid x') \quad \forall\, x, x'$. With this choice, the acceptance ratio (33) simplifies to

$$A(x', x) = \min\left(1, \frac{\pi(x')}{\pi(x)}\right)$$

**Consideration 2** From Equation (33), in order to obtain the acceptance ratio we need to compute ratio $\pi(x')/\pi(x)$, which requires the knowledge of $\pi$ up to an overall normalization factor. This is very important, since in many practical application $\pi(x)$ is in fact known only up to a normalization factor that is hard to compute.

**Consideration 3** Typically, proposal kernels $g(x' \mid x)$ are such that it is computationally cheap to sample from. A very common proposal kernel when dealing with binary variables (e.g. when a configuration $x$ is a string of $N$ bits, $x = (x_1, ..., x_N)$ with $x_n \in \{0, 1\}$) is defined by the following very easy to implement process (which should be used in line 3 in Alg. 4):

- sample a uniformly distributed number $i \in \{1, ..., N\}$
- flip the component $i$ of $x$, i.e. change $x_i$from 0 to 1 or viceversa.

Notice that this kernel is symmetric.

# 3 The Simulated Annealing Algorithm

In this section we describe Simulated Annealing, a very important and versatile algorithm that allows to perform optimization by means of Markov Chain Monte Carlo (MCMC). To this aim we will use the Metropolis algorithm and see how it can be extended to become a solver for optimization problems.

## 3.1 The Boltzmann distribution

Consider a discrete probability distribution $\pi(x)$. By the very definition of probability distribution (density), we have the following two properties

- $p(x) \geq 0 \ \forall x$. That is, $p(x)$ is non-negative.
- $\sum_x p(x) = 1$.That is, $p(x)$ is normalized (to 1).

Now let's restrict ourselves to the case where $p(x)$ is always strictly larger then zero, $p(x) > 0 \ \forall x$ (the general case where $p(x) = 0$ for some $x$ can be easily accommodated by restricting the configuration space). Then there exists a function $E(x)$ and a constant $Z$, such that we can express $\pi(x)$ as

$$p(x) = \frac{e^{-E(x)}}{Z}. \tag{34}$$

The normalization factor $Z$ is simply given by $Z = \sum_x e^{-E(x)}$. It is easy to see that this definition satisfies both properties of a well defined probability function.

- $E(x)$ is called *energy function*. It is usually easy to evaluate.
- $Z$ is called *partition function*. It is usually hard to compute since it typically involves the sum over a large number of configurations. Fortunately we don't have to worry about the hardness of computing $Z$ in MCMC algorithms (see Consideration 2 in Section 2.6).

Also, notice that for a given $p(x)$, the corresponding $E(x)$ is not unique, since the partition function can absorb any constant factor in the numerator. In fact, adding to $E(x)$ any constant term $C$, $E_2(x) = E(x) + C$, we obtain a new energy function that describes the same probability distribution. Therefore the energy function is defined only up to a constant term.

It is useful to consider the converse situation, as people in *statistical physics* often do. The starting point now is the *energy function* $E(x)$ and a parameter $T > 0$ is called the *temperature*. Also, usually one calls $\beta = \frac{1}{T}$ the *inverse temperature*. Given an energy function and a temperature, one can define the *Boltzmann probability distribution* $\pi(x)$ *as*

$$\pi_T(x) = \frac{e^{-\frac{1}{T}E(x)}}{Z_T}, \tag{35}$$

where as usual the partition function is defined so that it normalizes the distribution to one: $Z_T = \sum_x e^{-\frac{1}{T}E(x)}$.

The temperature $T$ is exactly the same quantity you measure with thermometers, but it may also be considered as a tuning parameter that changes the shape of the Boltzmann distribution, as we will now see. For very large temperature ($T \gg 1$), $\pi_T(x)$ becomes the uniform distribution (Fig. 7), giving to all configurations the same weight:

$$\pi_T(x) \approx \frac{1}{\sum_{x'} 1} \qquad \text{for } T \gg 1 \tag{36}$$

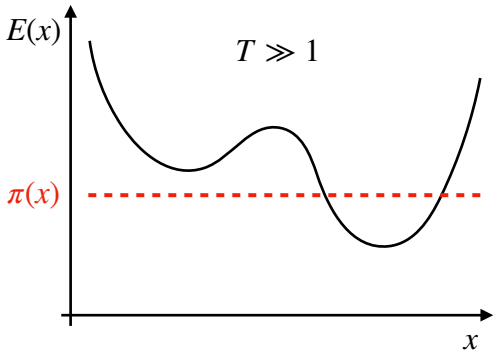


**Figure 7:** Black line: energy function $E(x)$. Red line: Boltzmann distribution $\pi_T(x)$ in the limit of high temperature $T$.

At intermediate values of the temperature ($T \approx 1$), the configurations with lower energies are preferred (i.e. the probability density is higher in correspondence of these configurations.) (Fig. 8).

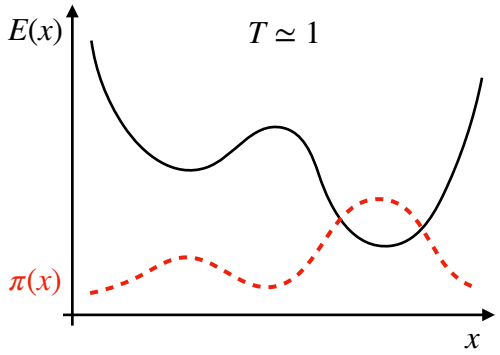


**Figure 8:** Black line: energy function $E(x)$. Red line: Boltzmann distribution $\pi_T(x)$ for intermediate values of the temperature $T$.

When the temperature is close to zero ($T \ll 1$) the distribution $\pi_T(x)$ is peaked around the smallest energy configuration (Fig. 9) called *ground state*, namely the *global minimum* of $E(x)$.

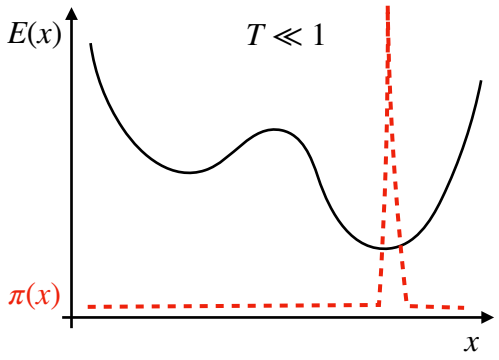


**Figure 9:** Black line: energy function $E(x)$. Red line: Boltzmann distribution $\pi(x)$ for low values of the temperature $T$.

**Note:** For any $T < +\infty$ the configurations with lower energy are favored, but typically there are *many more* high energy configurations than low energy ones, so while sampling from $\pi_T(x)$ you may almost always get high energy configurations: this is called *entropic effect*. Entropy is a measure of how many possible configurations your system may assume, and this is related to disorder. An everyday-life example of this effect is the following: your room may appear always messy, unless you put a lot of effort into tidying it up. This is because there are many ways for it to be messy and just a few ordered configurations. In the Boltzmann distribution the temperature $T$ tunes the balance between energy minimization (low $T$) and entropy maximization (high $T$).

## 3.2 Sampling from the Boltzmann distribution

We can sample from the Boltzmann distribution using Markov Chain Monte Carlo with the Metropolis algorithm

**Algorithm 5** Metropolis for Boltzmann sampling

**Require:** energy function $E$, temperature T
**Require:** stopping time $t_{max}$, initial configuration $x^0$
**Ensure:** a configuration sampled from the Boltzmann distribution $\pi_T$

1: $x \leftarrow x^0$
2: **for** $t = 1, ..., t_{max}$ **do**
3: $x' \leftarrow$ sample from $g(\bullet \mid x)$ ▷ *propose a move from $x$ to $x'$*
4: $\Delta E \leftarrow E(x') - E(x)$
5: $r \leftarrow \text{Uniform}(0, 1)$
6: **if** $r \leq \min(1, e^{-\Delta E/T})$ **then** ▷ *accept the move*
7: $x \leftarrow x'$
8: **end if**
9: **end for**
10: **return** x

So here you see that the important quantity is the energy difference $\Delta E$: if $\Delta E < 0$, i.e. the configuration $x'$ has lower energy than the configuration $x$, you accept the move with probability one (as you see from line 6 of the

pseudocode and considering that $e^{-\Delta E/T} < 1$ if $\Delta E < 0$). So you're preferring configurations with lower energy. On the other hand, if $\Delta E > 0$ (i.e. $x'$ has higher energy than $x$), you can still accept the move, but only with probability $e^{-\Delta E/T}$. The temperature parameter $T$ tunes the probability by which energy increasing proposals are accepted, from always ($T = +\infty$) to never ($T = 0$).

## 3.3 Optimization with the Metropolis algorithm

For some "objective / cost / loss / energy" function $E(x)$ (the name depends on the scientific field considered) defined on some space $\mathcal{X}$, consider the following optimization problem:

$$x^* = \min_{x \in \mathcal{X}} E(x).$$

We call $x^*$, the global minimum of $E(x)$, the *ground state* of the the system. According to our previous discussion in Section 3.1, we could solve the problem if we were able to sample from the zero-temperature ($T = 0$) Boltzmann distribution (as it is peaked on the lowest energy configuration). We are therefore tempted to use the following variation of the Metropolis algorithm:

**Algorithm 6** Zero-temperature Metropolis algorithm

**Require:** energy function $E$
**Require:** stopping time $t_{max}$, initial configuration $x^0$
**Ensure:** a candidate ground state.
1: $x \leftarrow x^0$
2: **for** $t = 1, ..., t_{max}$ **do**
3: $x' \leftarrow$ sample from the kernel $g(\bullet|x)$ ▷ *propose a move from $x$ to $x'$*
4: $r \leftarrow \text{Uniform}(0, 1)$
5: $\Delta E \leftarrow E(x') - E(x)$
6: **if** $\Delta E \leq 0$ **then** ▷ *only energy decreasing moves*
7: $x \leftarrow x'$
8: **end if**
9: **end for**

Here you see that the difference w.r.t. to the standard Metropolis algorithm is contained again in line 6 of the pseudocode: you accept the move only if $\Delta E \leq 0$. The $0T$ Metropolis algorithm is a *stochastic greedy heuristic* for solving optimization problems:

*stochastic*: the proposal $x'$ is stochastic and depends only on the current configuration $x$ (it is a Markov Chain);

*greedy*: only the moves that decrease (or not-increase) the energy are accepted. Since the moves are typically local ($x'$ close to $x$) this may not be optimal and it may get stuck in a local minimum;

*heuristic*: the algorithm is *not* guaranteed to solve the problem, but it may reach a good configuration

For hard combinatorial problem such as the Traveling Salesman Problem, this greedy strategy is very likely to fail, preforming premature optimization and returning some sub-optimal solutions. In the next section we present a much more powerful algorithm to deal with "rough landscape" problems.

## 3.4 Simulated Annealing

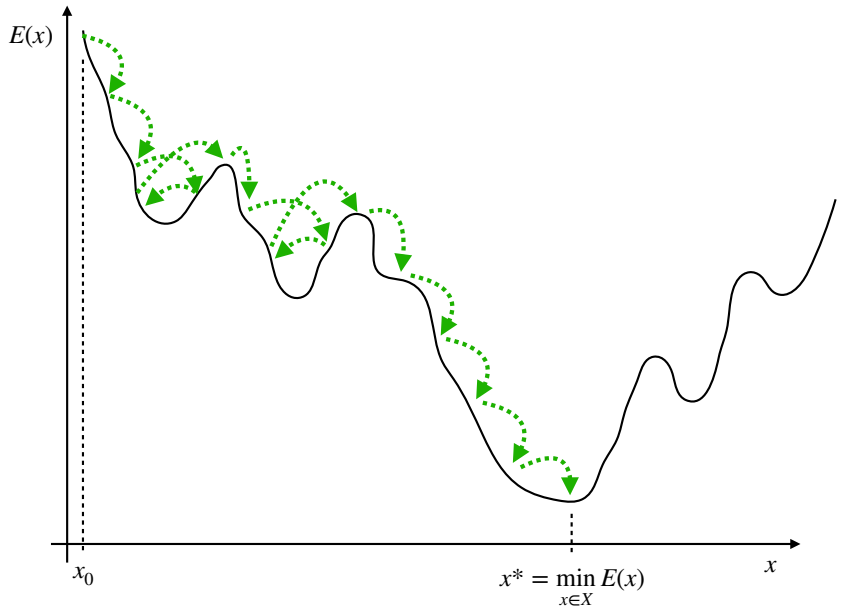


**Figure 10:** Example of a SA trajectory. A greedy MCMC algorithm would get stuck in the first local minimum valley, while SA is able to overcame the small barriers because it can accept moves that increase the energy while $T > 0$. The algorithm should *ideally* be tuned in such a way that the temperature reaches zero when sampling in proximity of the global minimum.

The idea behind the Simulated Annealing (SA) algorithm is that a MCMC simulation at *high* temperature is able to explore the state space without getting trapped by energy barriers. Since in an optimization problem one wants to ultimately find the ground state (global minimum), it may be convenient to *gradually lower the temperature* to drive the system towards regions of lower and lower energy. The temperature sets the balance between exploration and exploitation: we want to explore a lot at the start of simulation, and we want to exploit the collected information at the end.

We define a decreasing temperature schedule

$$T^0 \geq T^1 \geq T^2 \geq ... \geq T^t \geq ... \geq T^{t_{\max}} = 0$$

and we use the schedule inside our Metropolis algorithm. Ideally, if the schedule is slow enough, at time $t$ the algorithm will be roughly sampling from the Boltzmann distribution with temperature $T^t$, and as time goes by the sampling will track the changing shape of the distribution.

A possible choice for the temperature schedule is

$$T^{t+1} = T^t - \Delta T$$

for some constant $\Delta T$ that becomes a parameter of our SA algorithm. Many other different schedule could be considered. The algorithm is presented as Alg. 7.

**Algorithm 7** Simulated Annealing

**Require:** energy function $E$, temperature schedule $\{T^t\}_t$
**Require:** stopping time $t_{max}$, initial configuration $x^0$
**Ensure:** a candidate ground state.

$x \leftarrow x^0$
**for** $t = 1, ..., t_{max}$ **do**
  $x' \leftarrow$ sample from the kernel $g(\bullet \,|\, x)$
  $r \leftarrow \text{Uniform}(0, 1)$
  $\Delta E \leftarrow E(x') - E(x)$
  **if** $r \leq \min(1, e^{-\Delta E/T^t})$ **then** ▷ *accept. prob. depends on t*
    $x \leftarrow x'$
  **end if**
**end for**
**return** x

## 3.5 Application: The Traveling Salesman Problem

The Traveling Salesman problem TSP is a celebrated problem in combinatorial optimization, defined as it follows:

**TSP problem:** Given a list of $N$ cities and a matrix $d_{ij}$ of distances between each pair of them, find the *tour* which visits all the cities *just once* and has the *minimum total length* among all tours.

An example tour on 5 cities is $1 \to 2 \to 4 \to 3 \to 5 \to 1$. Given a problem with $N$ cities, one has to find the tour with *minimum total length* among $N!/N$ possible tours. The division by $N$ is due to the fact that for this problem it is not important from which point you start (the solution is invariant w.r.t. the starting point).

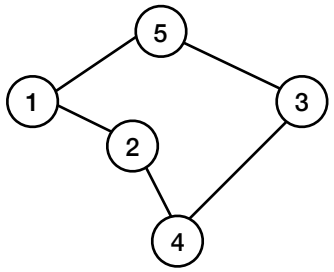


**Figure 11:** A simple instance of the TSP problem with $N = 5$. The lines between the dots represent the shortest of $\frac{5!}{5} = 24$ possible tours.

The problem is NP-complete, so we don't expect an efficient solver to exist for any problem instance. However, *typical instances* may be easy enough for a powerful solver such simulated annealing to reach a solution in a reasonable time. [2]

There are many different ways to represent a tour. For example, in programming languages with zero-based array indexing, you may want to represent a tour as a permutation of the indexes of the cities stored into an array

[2] The shortest tour among all UK pubs ($\simeq 50000$) was recently computed. Link.

$t$ of length $N$, such that $t[k]$ is the city visited at step $k$ and $k[0]$ is the city visited at step 0 and $N$ (the step index $k$ has values $0 \leq k \leq N-1$).

In this setting the cost function $E$ of the tour $t$ can be expressed as:

$$E(t) = \sum_{k=0}^{N-1} d_{t[k],t[(k+1)\%N]}$$

where $d_{t[k],t[k']}$ is the distance between the cities $t[k]$ and $t[k']$. So this is a sum over the distances between the adjacent cities of the chosen tour. The modulus operator $\%N$ indicates that the first and the last city must coincide in order to close the tour.

Note that this representation is slightly redundant, since $t$ can represent $N!$ possible permutations while there are only $N!/N$ different tours (the starting point does not matter). For example, the permutations corresponding to the tour in Fig. 11 are $\{[1,2,4,3,5],[2,4,3,5,1],\dots\}$; in other words the cyclic translations of any $t$ correspond to the same tour (you have $N$ of them).

The simplest *greedy algorithm* is the following:

**Algorithm 8** simplest greedy algorithm

1: Start from any city;
2: Add to tour the closest city not already in the tour;
3: Repeat line 2 until all $N$ cities are in the tour;
4: Go back to the initial city.

It turns out that the greedy algorithm yields highly suboptimal solutions.[3] We can do much better with SA, but first we have to introduce a more convenient formulation of the problem.

We associate to each potential edge between cities $(i,j)$ an occupation number $n_{ij}$; for convenience we use a symmetric matrix: $n_{ij} = n_{ji}$ (if city $i$ is connected with $j$, obviously also $j$ with $i$) and $n_{ii} = 0$ (non self-loops). Then we define

$$n_{ij} = \begin{cases} 1 & \text{if } (i,j) \text{ is in the tour} \\ 0 & \text{otherwise} \end{cases}$$

At any moment during the execution of the algorithm, the *candidate optimal tour* is given by

$$t = \{(i,j) : i < j \text{ and } n_{ij} = 1\}.$$

We can then formulate the TSP problem as an *integer optimization problem*:

$$\min_n \left\{ E(n) := \sum_{i<j} d_{ij} n_{ij} \right\} \tag{37}$$

where $d_{i,j}$ is the distance between city $i$ and city $j$ and such that the following constraints hold:

1. $n_{ij} \in \{0,1\}\ \forall i,j,\ i<j$;
2. $\sum_j n_{ij} = 2\ \forall i$; that means that each city is in the tour and it is visited only once (in practice, by summing the matrix $n_{i,j}$ over the city $j$ you are looking if this city is connected with others cities $i$: if the answer is positive, for each city $i$ to which $j$ is connected you get a summand of 2 because $n_{i,j}$ is symmetric);

[3] *Exact* solvers for the TSP are based on branch-and-cut or branch-and-bound approaches.

3. The configuration $\{n_{ij}\}$ is not a union of disjoint tours.

In general optimization problems, the set of configurations satisfying all of the constraint is called *feasible set*.

Now, we can make our choice for the Metropolis move, i.e. how to propose a new configuration starting from the previous one. Once chosen the Metropolis move, in the algorithm we will have to compare the energies of the previous and the new configurations. A convenient choice is the following. Let $(i, j)$ and $(k, l)$ be two distinct edges in the candidate tour.

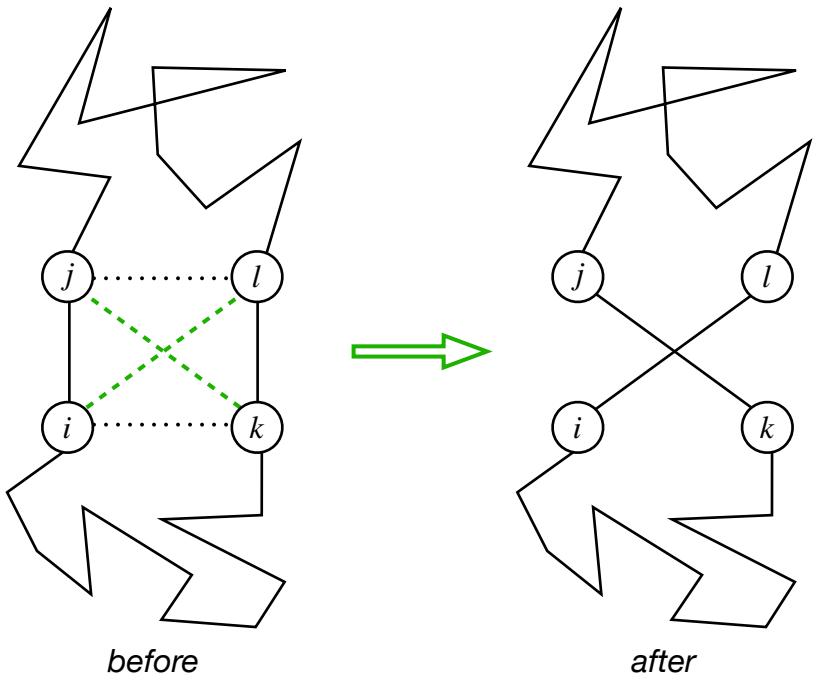


**Figure 12:** Example of a move involving the edges between the cities $i, j, k, l$.

In Fig. 12 we replaced $(i, j)$ and $(k, l)$ with $(i, l)$ and $(k, l)$ to obtain another tour. Notice that the other possible way of connecting $i, j, k, l$ (dotted lines) would yield disjoint subtours, therefore it is not admissible move. Let us call $n$ and $n'$ the configurations before and after the rearrangement respectively. The energy difference, as you can see from the energy defined in Eq.(37) is given by:

$$\Delta E = E(n') - E(n) = d_{ij} + d_{kj} - d_{ij} - d_{kl}.$$

The key observation now is that $\Delta E$ is easy to compute since it involves only a few terms, namely only the ones locally involved in the move. This fact makes a single Metropolis move computationally fast to compute and therefore the SA algorithm powerful.

# A Notions of Graph Theory

A graph $G = (V, E)$ is a mathematical structure defined by a vertex set $V$ and an edge set $E$. The vertex set in a graph with $N$ vertices is typically denoted by the set of the first $N$ integers $V = \{1, ..., N\}$. The elements of the edge for undirected graphs set are unordered couples of vertices. A tree $T = (V', E')$ in a graph $G = (V, E)$ is a subset of $G$ that doesn't contain any cycle (circuit that doesn't repeat any vertex, path with the first and the last vertices equal). A spanning tree in a graph G is a tree in a graph G that covers all the vertices of G ($V' = V$).