Chapter 3

Simulation

This chapter is devoted to the simulation of Markov Chains on computers. Simulating consists in generating a *trajectory* of the process which obeys the rules of the model.

It is understood that the principal purpose of generating trajectories is to perform statistics on them. But this topic will not be developed in detail during the course. Some examples of the applications of simulation are:

- visualize the behavior of some random system;
- obtain samples of the random variable $X(n)$ and estimate its distribution $\boldsymbol{\pi}_n$;
- obtain samples of the stationary distribution: this topic will be developped below;
- obtain samples of pairs of random variables $(X(n), X(m))$ and estimate their correlation; study the *spectral density* of the random process;
- obtain samples of *reaching times*, that is, the time it takes to enter a certain part of the state space; estimate probabilities of entering some subspace or another, depending on the initial state;
- obtain samples of a *total reward* (or total) accumulated along the trajectory.

The case of discrete-time chains is studied first in Section 3.1. Then the details of simulating continuous-time chains are discussed in 3.2. In Section 3.3, we show through an example how taking into account precisely the dynamics of the model allows to simulate Markov chains that cannot be handled by the basic methods. Finally, in Section 3.4, we study the problem of sampling from stationary distributions of Markov chains.

3.1 Simulation of Discrete-Time Markov Chains

The simulation of a DTMC consists in generating a finite sequence of states $(x[0], x[1], \ldots, x[N])$ which is distributed according to the process $\{X(n); n \in \mathbb{N}\}$ (more precisely, finite portions of it).

The data of the problem is: the transition matrix $\mathbf{P}$, and the initial probability distribution $\boldsymbol{\pi}_0$.

At the heart of the technique, there is the sampling of discrete distributions. In addition to $\boldsymbol{\pi}_0$, there are n discrete distributions to be considered, one per state of $\mathcal{E}$. If the state space is large or infinite, this may raise a practical problem: this will be addressed in Section 3.3. Let $\mathbf{p}_i$ denote the distribution corresponding to line i of the matrix $\mathbf{P}$. A random state drawn from this distribution corresponds to a jump from state i, according to the rules of the Markov chain.

Algorithm 3 describes the typical implementation of a simulation for DTMC. In this algorithm, the parameter N is the number of time steps to be simulated, in addition to the initial step. It is called the *horizon* or *duration* of the simulation. We present here the variant where this number is supposed to be known in advance. In some situations, this number is determined by the program itself based on a *stopping criterion*. We leave it to the reader to adapt the algorithm to this case.

The algorithm assumes the existence of a function `DiscreteSample`, with argument a discrete distribution, which returns a sample of this distribution. This function can typically be implemented using Walker's method (Algorithm 19 in Appendix B) or any other method. Note that this function is typically called a large number of times: it is therefore essential that it be optimized for speed.

Algorithm 3: Simulation of a DTMC

Data: A transition matrix $\mathbf{P}$, in the form of n distributions $\mathbf{p}_i$, $i \in \mathcal{E}$
Data: An initial distribution $\boldsymbol{\pi}_0$
Data: A time horizon $N \in \mathbb{N}$
Result: A sequence of $N+1$ states $x[i]$, $0 \leq i \leq N$
begin
  $x[0] \leftarrow$ `DiscreteSample`$(\boldsymbol{\pi}_0)$
  for n **from** *1* **to** N **do**
    $nextDist \leftarrow \mathbf{p}_{x[n-1]}$
    $x[n] \leftarrow$ `DiscreteSample`$(nextDist)$
    // insert here any processing on sample $x[n]$
  // insert here any processing on the sequence $x[0:N]$
return $x[0:N]$

In the text of the algorithm, comments indicate where the processing of the simulation result should occur. Processing may be statistical calculations, storing, printing or tracing or otherwise displaying the result. Depending on whether this processing involves only the last sample generated $x[n]$ or the complete sequence $x[0:N]$, it may be executed "on the fly" within the inner loop, or only at the end ("post-processing"). If no post-processing is necessary, it is possible to dispense with storing *all* values $x[n]$: only the current one is needed. This may result in important savings of memory, if N is a large number.

3.2 Simulation of Continuous-Time Markov Chains

The trajectory of a DTMC on a discrete space $\mathcal{E}$ is a piecewise-constant function of time, such as the one displayed in Figure 3.1. The trajectory changes states at discrete instants of time which form a random process $\{T_n; n \in \mathbb{N}\}$. Since this function of time is discontinuous, its value at the instants T_n has to be defined. The usual convention is to consider that the trajectory is *right-continuous*.

This is materialized by the black dots in Figure 3.1.

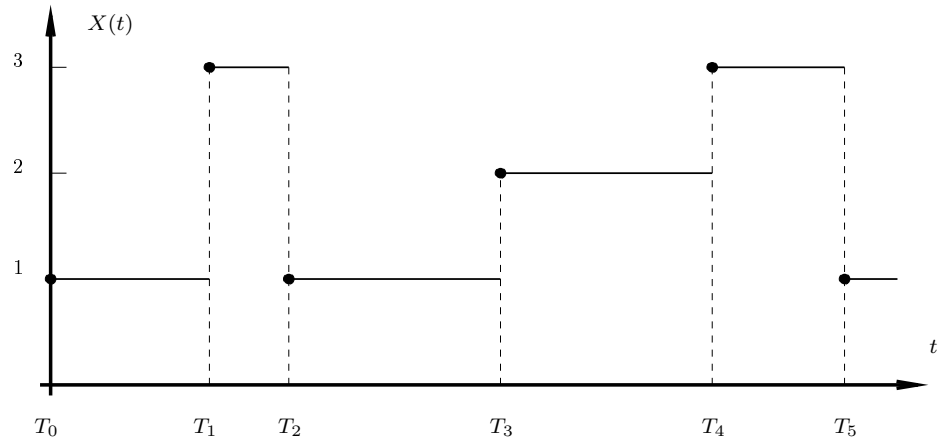


Figure 3.1: One trajectory of a 3-state CTMC

The lengths of the segments as well as the successions of states depend precisely on the infinitesimal generator of the chain. The next section is devoted to a better understanding of this relationship.

3.2.1 Trajectories and the construction of generators

The length of the constant segments of the trajectory are random variables. These are called *sojourn times* in the different states. We have the following theoretical result:

Theorem 3.1. *Consider a CTMC with generator* $\mathbf{Q}$ *over the discrete state space* $\mathcal{E}$*. Then:*

i/ the sojourn time in state i *is a random variable with exponential distribution with parameter* $\nu_i := |Q_{ii}| = -Q_{ii}$*; different sojourn times are independent random variables;*

ii/ the probability that the process jumps in state j *after its stay in state* i *is* Q_{ij}/ν_i*; the different jump events are independent from one another, and are independent on sojourn times.*

This result implies two "practical" characteristics of the process (which can be checked through simple statistics):

- The average sojourn time in state i is $m_i = 1/\nu_i$.
- The frequencies of jumps from i to j is $p_{ij} = Q_{ij}/\nu_i$.

Conversely, all processes which have the properties expressed in Theorem 3.1 are CTMCs. This provides a direct "construction" of the infinitesimal generator $\mathbf{Q}$ of a process, from a description of the rules of evolution. We provide two such constructions.

Theorem 3.2 (Construction #1)**.** *Consider a stochastic process in continuous time,* $\{X(t), t \in \mathbb{R}^+\}$*, having the following properties. When* X *enters state* i*:*

- *it stays in this state a random duration with exponential distribution with parameter τ_i;*
- *then jumps instantaneously in state $j \neq i$ with some probability p_{ij};*

and is such that jumps and durations are independent random variables.

Then the process $\{X(t), t \in \mathbb{R}^+\}$ is a CTMC which infinitesimal generator $\mathbf{Q}$ is:

$$Q(i,i) \;=\; -\tau_i \qquad Q(i,j) \;=\; \tau_i \; p_{ij} \;, i \neq j \;.$$

There is another interpretation of the evolution of Markov Chains, which is useful in certain situations where transitions are provoked by certain random "events".

Theorem 3.3 (Construction # 2)**.** *Consider a stochastic process in continuous time, $\{X(t), t \in \mathbb{R}^+\}$, having the following properties. When X enters state i:*

- *a potential transition to state j occurs after some random time Y_{ij} exponentially distributed with parameter ν_{ij}; it is possible that $\nu_{ij} = 0$, in which case $Y_{ij} = +\infty$;*
- *the process effectively stays in state i for a duration equal to $\min_j\{Y_{ij}\}$, then jumps to any state k which realizes the minimum;*

and is such that all potential durations Y_{ij} are independent random variables; it is also assumed that $\sum_j \nu_{ij} < \infty$, for all i.

Then the process $\{X(t), t \in \mathbb{R}^+\}$ is a CTMC which infinitesimal generator $\mathbf{Q}$ is:

$$Q(i,i) \;=\; -\sum_j \nu_{ij} \qquad Q(i,j) \;=\; \nu_{ij} \;, \quad i \neq j \;.$$

The fact that this second construction results in a CTMC is a consequence of the fact that the minimum of independent exponentially distributed random variables is again exponentially distributed. See Section 1.2.1.

Examples. Consider again the generator used as an example in Section 1.2.5:

$$\mathbf{Q} \;=\; \begin{pmatrix} -0.3 & 0.3 & 0 \\ 0.6 & -1.2 & 0.6 \\ 0.3 & 0.6 & -0.9 \end{pmatrix} \;.$$

The parameters corresponding to Construction #1 (Theorem 3.2) are:

$$\boldsymbol{\tau} \;=\; \begin{pmatrix} 0.3 \\ 1.2 \\ 0.9 \end{pmatrix} \;, \qquad \mathbf{P} \;=\; \begin{pmatrix} 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{2}{3} & 0 \end{pmatrix} \;.$$

Note that this matrix $\mathbf{P}$ is *not*, in general, one uniformized transition matrix of $\mathbf{Q}$.

Consider now the generator:

$$\mathbf{Q} = \begin{pmatrix} -2\alpha & \alpha & \alpha \\ 0 & -\beta & \beta \\ \gamma & 2\gamma & -3\gamma \end{pmatrix}$$

The parameters corresponding to Construction #1 are:

$$\boldsymbol{\tau} \;=\; \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} \;, \qquad \mathbf{P} \;=\; \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \\ \frac{1}{3} & \frac{2}{3} & 0 \end{pmatrix} \;.$$

3.2.2 Simulation Algorithms

As we have seen above, the trajectories of a CTMC are formed of a sequence of intervals where $X(t)$ is constant. Such a piecewise-constant function is conveniently represented as a sequence $\{(T_n, X_n); n \in \mathbb{N}\}$ where, $0 = T_0 < T_1 < \ldots$ and, by convention:

$$X(t) \;=\; X_n \;, \qquad \text{for all } t \in [T_n, T_{n+1}) \;.$$

Simulating the CTMC consists in computing a finite portion of such an infinite sequence.

As in the case of DTMC, a simulation runs for a certain *time horizon*. This number can be specified to the simulator, or be determined based on some stopping criterion.

Alternately, one may specify a certain number of transitions N to be simulated. In that case, the total duration of the simulation will not be known in advance.

The first simulation algorithm, Algorithm 4, is directly inspired from the interpretation in Theorem 3.1 and Construction #1. The basic random values are provided by the procedures `Exponential`, which returns samples of exponentially distributed random variables, and `DiscreteSample`, which returns samples of some discrete distribution. See Appendix B.2.1 and B.2.2 or B.2.3 for their description.

Another possibility is to use *uniformization* to convert the CTMC in a couple made of a Poisson process and a DTMC (see Theorem 1.18). A parameter ν is chosen which satisfies Condition (1.33). This defines a transition matrix $\mathbf{P}_\nu = \mathbf{I} + \nu^{-1}\mathbf{Q}$. The simulation consists in drawing at random, simultaneously, a Poisson process and the DTMC with transition matrix $\mathbf{P}$. This is Algorithm 5.

This uniformization method is close to one algorithm which is sometimes used, and which we describe now. Recall that from the definition of the infinitesimal generator, for every $i \neq j$,

$$\begin{aligned} \mathbb{P}(X(t+\delta t) = i \mid X(t) = i) \;&=\; 1 \;-\; \tau_i \delta t \;+\; o(\delta t) \\ \mathbb{P}(X(t+\delta t) = j \mid X(t) = i) \;&=\; Q_{ij}\delta t \;+\; o(\delta t) \;. \end{aligned}$$

One idea is then to chose δt (very) small, and calculate successive values of $X(m\delta t)$, $m = 0, 1, 2, \ldots$. Given that $X(m\delta t) = i$:

- draw $X((m+1)\delta t)$ according to the discrete distribution

$$\mathbb{P}(Y = j) \;=\; \delta t Q_{ij}, i \neq j, \qquad \mathbb{P}(Y = i) \;=\; 1 - \delta t \tau_i \;.$$

An equivalent method which may be slightly faster is:

- draw a sample B of a Bernoulli distribution with parameter $\tau_i \delta t$, that is, such that $\mathbb{P}(B = 1) = \tau_i \delta t$ and $\mathbb{P}(B = 0) = 1 - \tau_i \delta t$;
- if $B = 0$, set $X((m+1)\delta t) \leftarrow i$;

Algorithm 4: Simulation of a CTMC

Data: An infinitesimal generator $\mathbf{Q}$, in the form of transition rates τ_i, $i \in \mathcal{E}$, and distributions $\mathbf{p}_i$, $i \in \mathcal{E}$
Data: An initial distribution $\boldsymbol{\pi}_0$
Data: A time horizon $T \in \mathbb{R}$
Result: A number of events N
Result: A sequence of $N+1$ time instants $T[i]$, and states $x[i]$, $0 \leq i \leq N$
begin
  $T[0] \leftarrow 0$
  $x[0] \leftarrow \texttt{DiscreteSample}(\boldsymbol{\pi}_0)$
  $N \leftarrow 0$
  repeat
    $N \leftarrow N+1$
    $sojournTime \leftarrow \texttt{Exponential}(\tau_{x[N-1]})$
    $nextDist \leftarrow \mathbf{p}_{x[N-1]}$
    $T[N] \leftarrow T[N-1] + sojournTime$
    $x[N] \leftarrow \texttt{DiscreteSample}(nextDist)$
    // insert here any processing on sample $T[N], x[N]$
  until $T[N] \geq T$;
  // insert here any processing on the sequences $T[0:N]$, $x[0:N]$
return N, $T[0:N]$, $x[0:N]$

Algorithm 5: Simulation of a CTMC through uniformization

Data: A uniformized infinitesimal generator $\mathbf{Q}$, in the form of a rate ν and n transition distributions $\mathbf{p}_i^{\nu}$, $i \in \mathcal{E}$
Data: An initial distribution $\boldsymbol{\pi}_0$
Data: A time horizon $T \in \mathbb{R}$
Result: A number of events N
Result: A sequence of $N+1$ time instants $T[i]$, and states $x[i]$, $0 \leq i \leq N$
begin
  $T[0] \leftarrow 0$
  $x[0] \leftarrow \texttt{DiscreteSample}(\boldsymbol{\pi}_0)$
  $N \leftarrow 0$
  repeat
    $N \leftarrow N+1$
    $sojournTime \leftarrow \texttt{Exponential}(\nu)$
    $nextDist \leftarrow \mathbf{p}^{\nu}_{x[N-1]}$
    $T[N] \leftarrow T[N-1] + sojournTime$
    $x[N] \leftarrow \texttt{DiscreteSample}(nextDist)$
    // insert here any processing on sample $T[N], x[N]$
  until $T[N] \geq T$;
  // insert here any processing on the sequences $T[0:N]$, $x[0:n]$
return N, $T[0:N]$, $x[0:N]$

- if $B = 1$, then draw $X((m+1)\delta t)$ according to the discrete distribution

$$\mathbb{P}(Y = j) \;=\; \frac{Q_{ij}}{\tau_i} \;.$$

This algorithm has two drawbacks. First, it gives only an *approximate* trajectory. The sequence of states obtained does not have the exact distribution of the CTMC with generator $\mathbf{Q}$. Second, it performs many *unnecessary* computations: since δt has to be chosen very small, the "jumps" from state i are very often into state i. The first problem is corrected by replacing the *constant* duration δt with an *exponentially distributed* random value with average duration δt. This is actually what does the uniformized Algorithm 5. The second problem is corrected by "skipping" the useless jumps into the same state, by simulating directly the sojourn times in the different states. This is what Algorithm 4 does. In summary, this algorithm should **never** be used, unless there is a very good reason for not using algorithms 4 or 5.

3.3 Structure-based simulation

When the state space $\mathcal{E}$ is very large, or even infinite, the simulation algorithms proposed in the previous section cannot be used. Indeed, as they are written, they need as input matrices and vectors which represent a memory size of the order of $\#\mathcal{E}$. The assumption of this section is that such objects do not fit into the memory of the computer.

On the other hand, what the algorithms produce is a trajectory, that is, a sequence of T states, where T is the time horizon (or number of transitions). It is common that the storage in the memory of any state requires a modest amount of space. So, even though the *state space* cannot be stored, it is quite possible to store *trajectories*. In the rest of this section, we illustrate through an example how to simulate such a system. The example will also be used to illustrate the technique of "backwards simulation" in Section 3.4.2.

3.3.1 A dynamic Ising model

The following model is inspired from the well-known Ising model of Statistical Mechanics in Physics. It is taken from [20].

Consider a set of P "particles". Each of them has a state, which is an element of $\{-1, +1\}$. The state of particle $k \in \{1, \ldots, P\}$ is denoted with σ_k. The vector $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_P)$ denotes therefore the state of the system of particles. The state space of this system is $\mathcal{E} = \{-1, +1\}^P$; it has 2^P elements.

The particles are assumed to be arranged in some geometric configuration. Each system state has an *energy* defined as:

$$E(\boldsymbol{\sigma}) \;=\; -\sum_{i,j} A_{ij}\sigma_i\sigma_j \;-\; \sum_i B_i\sigma_i \;, \tag{3.1}$$

where the numbers A_{ij} are interpreted as coupling coefficients and B_i represent the intensity of an external field force. The intensity of the coupling typically depends on the distance between the particles. In some situations, it is non-zero only within a certain range: $A_{ij} = 0$ if $d(i, j) > R$.

Let us specify the dynamics of the system. Time is discrete. Given that the current state is $\boldsymbol{\sigma}$, the next state is determined by applying the following steps:

- select the set $\mathcal{C}$ of particles which are candidate for a change of individual state; two variants are proposed here, and other ones may be reasonable, depending on the assumptions of the model.

  Rule 1: choose one particle, uniformly at random on the set $\{1, \ldots, P\}$ (this is the rule used in [20], where it is called the "Glauber dynamics");

  Rule 2: choose each particle, independently with some probability p.

- for each particle k in the set $\mathcal{C}$, compute the difference of energy between candidate states:

  $$\boldsymbol{\sigma}_k^+ = (\sigma_1, \ldots, \sigma_{k-1}, +1, \sigma_k, \ldots, \sigma_P), \qquad \boldsymbol{\sigma}_k^- = (\sigma_1, \ldots, \sigma_{k-1}, -1, \sigma_k, \ldots, \sigma_P),$$

  that is:

  $$\Delta_k E(\boldsymbol{\sigma}) \;=\; E(\boldsymbol{\sigma}_k^+) - E(\boldsymbol{\sigma}_k^-) \;.$$

  Then set the value of σ_k to:

  $$+1 \text{ w.p. } \frac{e^{-\beta \Delta_k E(\sigma)}}{e^{-\beta \Delta_k E(\sigma)} + e^{\beta \Delta_k E(\sigma)}}, \quad \text{and } -1 \text{ w.p. } \frac{e^{\beta \Delta_k E(\sigma)}}{e^{-\beta \Delta_k E(\sigma)} + e^{\beta \Delta_k E(\sigma)}} \;.$$

  The parameter $\beta \geq 0$ is related to the temperature of the system.

When P is larger than 40, say, the state space is too large to be stored in the memory of current computers. However, since the state of the system can be coded on P bits, it is perfectly possible to generate and store trajectories. Algorithm 6 does this.

The algorithm calls two unspecified procedures. The function `Initialconfiguration()` returns the initial configuration. Depending on the purpose of the simulation, this configuration may be one regular arrangement (all individual states to $+1$, or to -1, or one geometric zone with $+1$ and the rest with -1), or one random arrangement, with each possible individual initial state randomly chosen, say, with equal probability.

The function `ChooseCandidates()` implements Rule 1 or Rule 2 above.

Let us now estimate the algorithmic complexity of this simulation. Storing the data requires *a priori* $O(P^2)$ memory units, principally to store the coupling matrix $\mathbf{A}$. This amount of storage may be reduced if the matrix is sparse. This situation occurs typically if the interaction has a short range, see above. The storage can be further reduced if the interaction obeys some regular rule, such as for instance: $A_{ij} = \mathbf{1}_{\{d(i,j) \leq R\}}$ or $A_{ij} = \gamma^{d(i,j)}$. In that case, the storage of $\mathbf{A}$ can be dispensed with altogether, and the simulation algorithm just needs the formula that computes A_{ij} when needed. The same observations apply to the vector B.

In order to evaluate the computation time required by the simulation, observe that for each time instant, a certain number of particles are selected, and for each of them, computations are performed. Under Rule 1, exactly one particle is selected. Under Rule 2, the number of particles selected is a random number, resulting from P independent random binary choices (select or not select). The distribution is therefore binomial, with average $p \times P$.

Algorithm 6: Simulation of the dynamic Ising model

Data: The number of particles P, the temperature parameter β
Data: Coupling coefficients A_{ij}, field coefficients B_i
Data: A time horizon $N \in \mathbb{N}$
Result: A sequence of $N+1$ states $x[i]$, $0 \leq i \leq N$
begin
  $x[0] \leftarrow$ `InitialConfiguration()`
  for n **from** *1* **to** N **do**
    $\mathcal{C} \leftarrow$ `ChooseCandidates()`
    foreach $k \in \mathcal{C}$ **do**
      compute $\delta \leftarrow \Delta_k E(x[n-1])$
      $p \leftarrow e^{-\beta\delta}/(e^{-\beta\delta} + e^{\beta\delta})$
      $U \leftarrow$ `Unif`$([0,1])$
      if $U \leq p$ **then**
        $x[n]_k \leftarrow +1$
      else
        $x[n]_k \leftarrow -1$
    foreach $k \notin \mathcal{C}$ **do**
      $x[n]_k \leftarrow x[n-1]_k$
    // insert here any processing on sample $x[n]$
  // insert here any processing on the sequence $x[0:N]$
return $x[0:N]$

For each selected particle, the value of $\Delta_k E(\boldsymbol{\sigma})$ must be computed, $\boldsymbol{\sigma}$ being the current state. Simple manipulations lead to the formula:

$$\Delta_k E(\boldsymbol{\sigma}) \;=\; -\,2\left(\sum_{i\neq k}(A_{ik}+A_{ki})\sigma_i \;+\; B_k\right)\;. \tag{3.2}$$

The number of operations needed to evaluate this is $O(P)$, with possible reductions if $\mathbf{A}$ is sparse, see above. The rest of the computations is done in constant time, that is, independently from the value of P or N. Finally, the total computational time for simulating N steps of this model is $O(NP)$ for Rule 1, and $O(pNP^2)$ for Rule 2. In the case of short-range interactions, this can be reduced to $O(N)$ and $O(pNP)$ respectively.

As a conclusion for this paragraph, we summarize some observations, obtained from this particular example but which turn out to have some degree of generality:

- Markov models are easily constructed by putting together a number of smaller models.

  In this Ising model, each particle can be considered as a two-state model. The resulting state space has a size that grows exponentially with the number of individual models.

- The transitions of such a model usually result from some mix of individual transitions (also called "local" transitions) with some sort of global coupling.

  In the present model, coupling occurs through the energy function. Another type of coupling occurs in the selection of particles which can change state at each step: particles can be somehow synchronized as in Rule 2.

- The dynamics of the system are formally and rigorously defined in a step-by-step, sequential description of random choices and events. This description is **customary** for an unambiguous specification of the model. It allows to construct the elements of the transition matrix $\mathbf{P}$ or the generator $\mathbf{Q}$. Last but not least: it is suitable for programming in a simulator.

- The computation time needed to simulate such a model grows moderately as a function of the number of individual models. At any rate, it is orders of magnitudes smaller than the size of the state space or the number of potential transitions from any state.

  This is particularly striking for Ising's model under Rule 2. Indeed, there is a positive probability to reach any state from any other at each step. In other words, the transition matrix is a full matrix! However, the simulation time is proportional to the square of the number of individual particles, because calculating transitions involves evaluating pairwise interactions. This relatively low complexity results from the "factoring" of transition probabilities through the sequential choices. This is analogous to the situation represented in Figure 3.2: in this decision tree, there is a certain (large) number of outcomes, each with some distinct probability. Choosing one outcome at random involves however a comparatively small number of random choices.

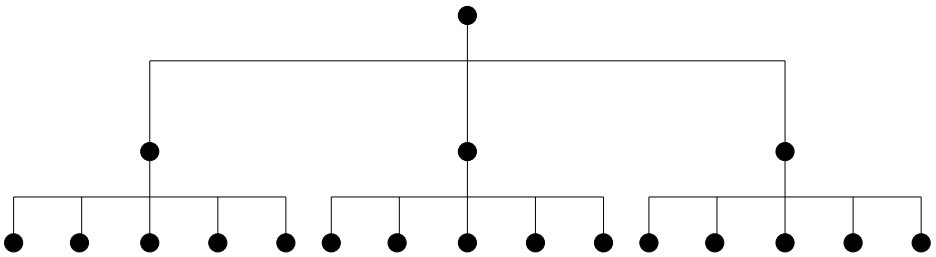


Figure 3.2: A decision tree

3.4 Sampling from stationary distributions

In some applications, one is interested in obtaining samples of the stationary distribution of some given Markov chain, through simulation.

When the probability transition matrix of the chain is given, two methods exist for this: the "forward" method and the "backwards" method. We shall describe them for DTMCs.

In other applications, a distribution of interest is given, but not a Markov chain. The challenge is then to define a Markov chain and its transition matrix, such that the given distribution is the stationary distribution of the chain. Samples of this distribution are then obtained by simulation. Methods of this category are called *Monte-Carlo Markov chains* (MCMC) methods.

3.4.1 Forward simulation

The forward simulation method consists in using Algorithm 3 with the time horizon N "large enough" so that the distribution of $x(N)$ (the algorithm produces samples of this distribution) is close enough to the stationary distribution. The question is how exactly to choose N.

Theorem 1.6 guarantees that this idea is valid, if the chain is irreducible and aperiodic (ergodic). In addition, Equation (1.16) says that the distance between $\boldsymbol{\pi}_N$ and the stationary distribution $\boldsymbol{\pi}$ decreases geometrically, and identifies the modulus of the *second eigenvalue* $|\lambda_2|$ as the rate of decrease. Assume that a bound on this value is known: let ρ be such that $|\lambda_2| < \rho < 1$. Then, according to (1.16), we have:

$$||\boldsymbol{\pi}_N - \boldsymbol{\pi}||_\infty \ \leq \ C\ \rho^N \ ,$$

for some constant C that, we assume also, can be computed. Under these conditions, it is possible to determine $N(\epsilon)$ such that the distance is less than ϵ: it suffices to take:

$$N(\epsilon) \ = \ \left\lceil \frac{\log(\epsilon/C)}{\log(\rho)} \right\rceil \ .$$

Clearly, the smaller is $|\lambda_2|$, the faster the convergence. The distance $1 - |\lambda_2|$ is called the *spectral gap* of the transition matrix. Unfortunately, it is often very difficult to obtain the value of $|\lambda_2|$ or even a good upper bound ρ (or, equivalently, a lower bound on the spectral gap).

In practice, one tries to determine empirically a good value of N, which makes an acceptable compromise between the duration of the simulation and the precision of the distribution. For instance, using some preliminary runs, one visualizes the empirical distribution of some random

variable involved in the system. If this distribution does not seem to change much when N is increased, one concludes that the system "has become stationary", and the value of N can be fixed.

3.4.2 Backward simulation

The method of "backward simulation" allows to obtain samples of the *exact* stationary distribution. We present here this technique, following [20] and [27, 28].[1] Is related to the notion of *coupling* of trajectories.

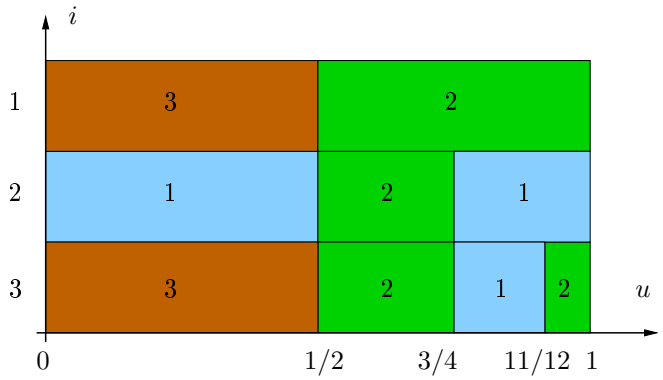
We are going to illustrate the idea through a very small example. Consider the 3-state DTMC with the following transition matrix:

$$\mathbf{P} \;=\; \begin{pmatrix} 0 & 1/2 & 1/2 \\ 3/4 & 1/4 & 0 \\ 1/6 & 1/3 & 1/2 \end{pmatrix} \;. \tag{3.3}$$

When one simulates the evolution of this Markov Chain with for instance Algorithm 3, it is necessary to obtain samples of the transition distribution corresponding to the current state i. This can always be obtained by constructing a function $\varphi_i(u)$, depending on a real parameter $u \in [0,1]$, such that if $U \sim \texttt{Unif}([0,1])$, then:

$$\mathbb{P}(\varphi_i(u) = j) \;=\; p_{ij} \;.$$

Such a construction is always possible: for instance with the naive Algorithm 17 in Appendix B.2.2. Since there is such a function for each state, we have actually a function $\Phi : [0,1] \times \mathcal{E}$ defined as: $\Phi(u,i) = \varphi_i(u)$. For any fixed u, this function realizes the mapping on states: $\varphi^u(i) : \mathcal{E} \to \mathcal{E} : i \mapsto \Phi(u,i)$.



$$\varphi^u \;=\; \begin{cases} (3,1,3) & \text{if } 0 < u \le 1/2 \\ (2,2,2) & \text{if } 1/2 < u \le 3/4 \\ (2,1,1) & \text{if } 3/4 < u \le 11/12 \\ (2,1,2) & \text{if } 11/12 < u \le 1. \end{cases}$$

Figure 3.3: One possible mapping $\Phi(u,i)$ for the matrix $\mathbf{P}$ in (3.3)

Figure 3.3 represents such a mapping for matrix $\mathbf{P}$. The mapping φ^u obtained for the different values of u is also listed, represented as the vector $(\varphi^u(1), \varphi^u(2), \varphi^u(3))$.

Given a function $\Phi(\cdot,\cdot)$, Algorithm 3 can be implemented as in Algorithm 7.

Except for the choice of the initial state, the unique source of randomness in this algorithm is the calls to the function $\texttt{Unif}([0,1])$. Assume that during the execution of the algorithm, the sequence

[1] This "perfect simulation" method has been implemented in the freely available software Psi2: `http://www-id.imag.fr/Logiciels/psi/`.

Algorithm 7: Forward simulation of a DTMC using random mappings

Data: A transition matrix $\mathbf{P}$, in the form of a mapping $\Phi : [0,1] \times \mathcal{E} \to \mathcal{E}$
Data: An initial distribution $\boldsymbol{\pi}_0$
Data: A time horizon $N \in \mathbb{N}$
Result: A sequence of $N+1$ states $x[i]$, $0 \leq i \leq N$
begin
  $x[0] \leftarrow$ `DiscreteSample`$(\boldsymbol{\pi}_0)$
  for n **from** *1* **to** N **do**
    $U \leftarrow$ `Unif`$([0,1])$
    $x[n] \leftarrow \Phi(U, x[n])$
    // insert here any processing on sample $x[n]$
  // insert here any processing on the sequence $x[0:N]$
return $x[0:N]$

of random numbers $(U_1, U_2, \ldots)$ happens to be $(0.3, 0.8, 0.9, 0.6, 0.2, \ldots)$. The successive sequence of mappings φ^{U_j}, $j = 1, \ldots, 5$, is represented in Figure 3.4 (top). The trajectories obtained for each possible value of the initial state are also represented. The same construction for the sequence $(0.2, 0.3, 0.8, 0.9, 0.6, \ldots)$ is represented in Figure 3.4 (bottom).

In both situations, the trajectories issued from all different initial states become the same after a certain time: $\tau = 4$ in the first situation, $\tau = 3$ in the second one. One says that trajectories are *coupled* and τ is the coupling time. After coupling has occured, the trajectory does not depend on the initial state. Since this is also one characteristic of the stationary distribution (see Theorem 1.6 for ergodic chains), a natural idea is that after coupling has occured, the system should have reached stationarity. However, this is not true in general: there is no reason why the distribution of $X(\tau)$ (the state of the system at the coupling time) should be $\boldsymbol{\pi}$. In addition, there is no reason why, in general, the coupling time τ exists.

In the example we have used, τ is finite with probability 1, because there exists one mapping that sends every state onto state 2 (for $u \in [1/2, 3/4)$). This lucky situation does not always occur.

Propp and Wilson have observed in [20] that when occurs *from the past*, the distribution of the state at the time of coupling is indeed the stationary distribution. We now explain this concept.

Since the DTMC is homogeneous (see Definition 1.1), the distribution of a piece of trajectory: $(X_m, X_{m+1}, \ldots, X_{m+n})$, conditioned on the value of the state X_m, does not depend on m. In particular, the conditional distributions of $(X_0, X_1, \ldots, X_n)$ and $(X_{-n}, X_{-n+1}, \ldots, X_0)$ are the same. This last trajectory can be seen as "starting in the past" at time $t = -n$ and being observed at time 0.

Coupling from the past consists in finding the closest time in the past such that all trajectories originating from there couple at time $t = 0$. This is what does Algorithm 8. The algorithm also returns the coupling time: this is sometimes useful for performing statistics and assessing the efficiency of the method. Observe the use of the temporary array `new_termState`: it is essential that the previous terminal states be kept unchanged in the memory until all new ones are computed.

The execution of the algorithm is illustrated in Figure 3.5, where it is assumed that the sequence of random numbers U is $(0.3, 0.8, 0.9, 0.2, \ldots)$. Coupling occurs after $\tau = 4$ steps. Steps 1, 2 and 4 of the algorithm are displayed. The set of "terminal states" observed at time 0 is highlighted, as

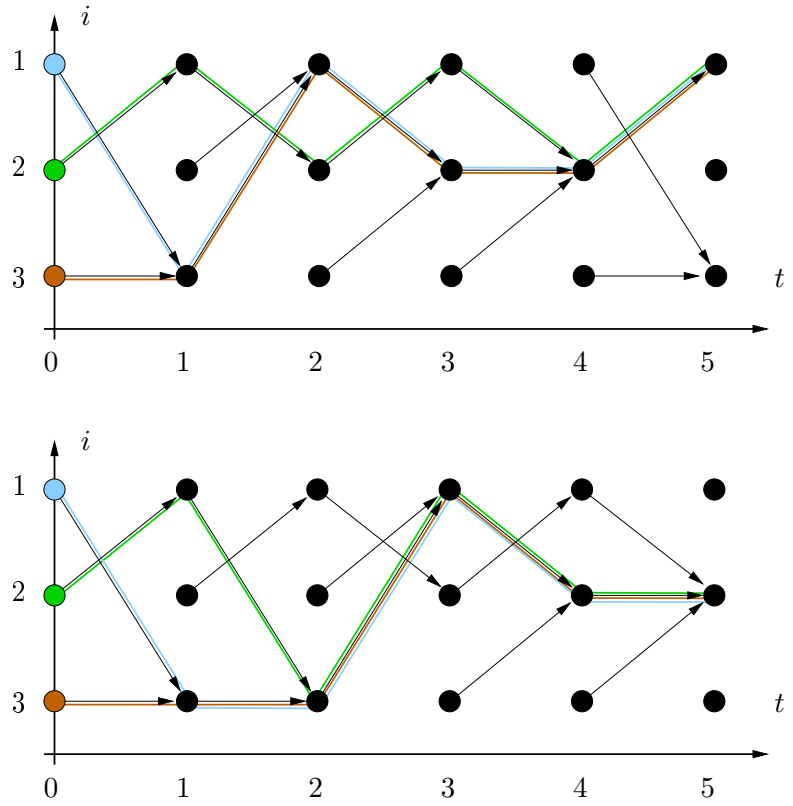


Figure 3.4: Mappings and trajectories for the example

Algorithm 8: Backward simulation of a DTMC

Data: A transition matrix $\mathbf{P}$, in the form of a mapping $\Phi : [0,1] \times \mathcal{E}$
Result: A random state of $\mathcal{E}$
Result: A coupling time τ
begin
  foreach $x \in \mathcal{E}$ **do**
    $\texttt{termState}[x] \leftarrow x$
  $\tau \leftarrow 0$ **repeat**
    $\tau \leftarrow \tau + 1$
    $U \leftarrow \texttt{Unif}([0,1])$
    foreach $x \in \mathcal{E}$ **do**
      $\texttt{new\_termState}[x] \leftarrow \texttt{termState}[\Phi(U, x)]$
    $\texttt{termState} \leftarrow \texttt{new\_termState}$
  until *all* $\texttt{termState}[x]$ *are equal*;
return *any* $\texttt{termState}[x]$, τ

well as the trajectories leading to these states.

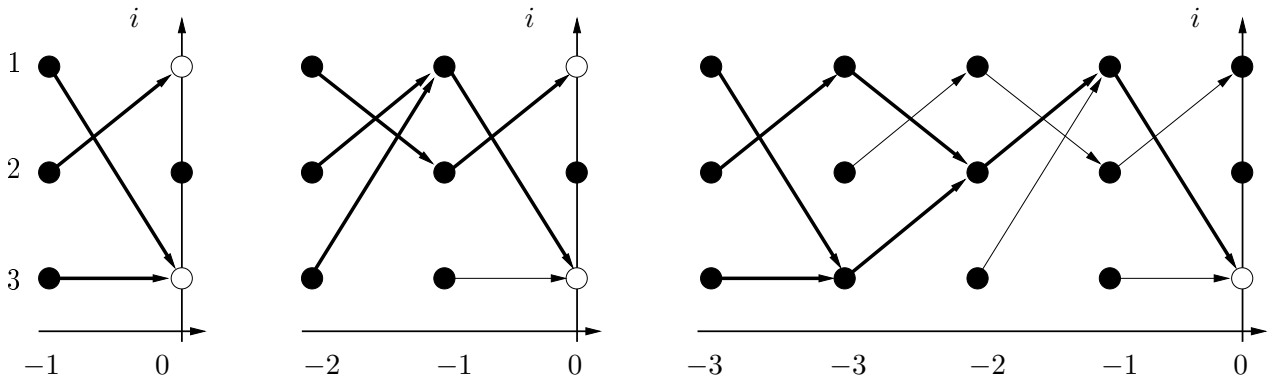


Figure 3.5: Mappings and trajectories for the example

The following results guarantees that samples returned by Algorithm 8 are distributed according to the stationary distribution. There is however no guarantee that the algorithm will stop in general. This depends on the choice of the mapping Φ, and it has to be checked case-by-case.

Theorem 3.4. *Assume that the Markov chain is ergodic and that Algorithm 8 stops with probability 1. Then the sample returned by it is distributed according to* $\boldsymbol{\pi}$.

The principal advantage of Algorithm 8 is that it returns samples of the exact stationary distribution, not of an approximation. For this reason, the method is called "exact sampling", and sometimes "perfect sampling" which is perhaps too strong.

The principal drawback is that the amount of memory it needs is at least the size of the state space $\mathcal{E}$. Indeed, the algorithm needs to store the vector `terminalState`. We see below cases where this vector can be dispensed with. The implementation of the mapping Φ may also require a memory proportional to $|\mathcal{E}|$. In that case, "structure based" methods as in Section 3.3 are useful to reduce memory consumption.

Another possible problem with the algorithm is that the mapping Φ must be correctly chosen, so that: 1) coupling occurs almost surely, and as fast as possible, and 2) its calculation does not need too much computational time. Reference [28] discusses how to address this last point, using Walker's method (described in Appendix B.2.3).

Monotone systems. In the remainder of this section, we consider that the state space is equipped with a partial order, denoted with $\prec$. Assume also for simplicity that there exist two states, σ^+ (the maximal element) and σ^- (the minimal element) such that: $\forall \sigma \in \mathcal{E}$, $\sigma^- \prec \sigma \prec \sigma^+$.[2] Assume that some mapping $\Phi(\cdot,\cdot)$ can be constructed such that

$$\forall u \in [0,1],\ \forall (i,j) \in \mathcal{E} \times \mathcal{E} \qquad i \prec j \quad \Longrightarrow \quad \Phi(i,u) \prec \Phi(j,u)\ . \tag{3.4}$$

The mapping Φ is *monotone* in the sense that it respects the partial order of states.

[2] In general, there are *sets* of minimal and maximal elements. Most of the principles discussed in this section can be generalized to this situation.

Consider the forward simulation of the Markov chain with such a mapping, such that the initial state is σ^-, σ^+ or any $\sigma \in \mathcal{E}$. Then because of (3.4), at each step n of the simulation, the three simulated states are such that:

$$\mathtt{x}^-[n] \prec \mathtt{x}[n] \prec \mathtt{x}^+[n] .$$

If at some step τ it happens that $\mathtt{x}^-[\tau] = \mathtt{x}^+[\tau]$, then necessarily, $\mathtt{x}[\tau] = \mathtt{x}^+[\tau]$ for all possible initial state σ. In other words, with the sequence of random variables $(U_1, U_2, \ldots)$ which led to the situation $\mathtt{x}^-[\tau] = \mathtt{x}^+[\tau]$, any set of trajectories issued from different initial states at time $-\tau$ must have coupled at time 0. Trajectories starting at any time $t \leq -\tau$ will have the same property. In order to check that coupling has occured, it is therefore sufficient to simulate *only two* trajectories: the ones corresponding to the two extremal elements.

This idea applies as well to coupling from the past. So, in principle, it is enough to simulate the action of the transition mapping on σ^- and σ^+ for τ units of time. In formulas, this provides the random state:

$$\sigma \;=\; \Phi(U_1, \Phi(U_2, \ldots, \Phi(U_{\tau-1}, \Phi(U_\tau, \sigma^+)) \ldots)) \;=\; \Phi(U_1, \Phi(U_2, \ldots, \Phi(U_{\tau-1}, \Phi(U_\tau, \sigma^-)) \ldots)) .$$

The problem is that the value of τ is not known in advance. The idea is then to increment the duration of the simulation, and simulate *forward* until coupling at time 0 is observed. But for each new value of n, the entire simulation must be done: it is not possible to reuse the result of the simulation on $n-1$ steps of time to get the result of the simulation on n steps. This was done in Algorithm 8 by storing the result in `termState`. We cannot do it anymore since we want to eliminate this large array. Incrementing the duration of the simulation by one turns out to be excessively costly. The good thing to do is actually to *double* the duration of the simulation, not to increment it by one.

This results in Algorithm 9. The algorithm does longer and longer sub-simulations, until coupling. When doubling the duration of the sub-simulation, so that $-2T \leq t \leq 1$, is is important that the transitions with $T \leq t \leq 1$ be computed with the *same* value of U. Otherwise, there is a risk that the samples be biased. This is why the algorithm stores the samples of $\mathtt{Unif}([0,1])$ so that they can be reused. This comes at the cost of some memory, which is linear as a function of τ. This is acceptable if the model is such that τ is relatively small, as compared with the size of the state space. In practice, since random number generators usually work with a *seed* (which is sometimes the last value generated), it suffices to store these values at the start of each sub-simulation, and not all values.

The dynamic Ising's model described in Section 3.3.1 is a monotone system. Define the partial order on states as $\boldsymbol{\sigma} \prec \boldsymbol{\sigma}'$ iff $\sigma_i \leq \sigma'_i$ for all i. The extremal elements of this order are $\boldsymbol{\sigma}^- = (0, \ldots, 0)$ and $\boldsymbol{\sigma}^+ = (1, \ldots, 1)$. Assume that the coupling coefficients A_{ij} are nonnegative. According to Equation (3.2), we have for every particle k:

$$\boldsymbol{\sigma} \prec \boldsymbol{\sigma}' \quad \Longrightarrow \quad \Delta_k E(\boldsymbol{\sigma}) \geq \Delta_k E(\boldsymbol{\sigma}') \quad \Longrightarrow \quad p_k := \mathbb{P}(\sigma_k \to 1) \leq p'_k := \mathbb{P}(\sigma'_k \to 1) .$$

Then, Algorithm 6 already implements a mapping function Φ which preserves the order. Compare two runs of it with different initial states. At some step, if the two current states are such that $\boldsymbol{\sigma} \prec \boldsymbol{\sigma}'$, then the particles chosen as candidates for a move are exactly the same since they result from the same call to `ChooseCandidates()`: this choice is already coupled. Next, for eack k candidate,

Algorithm 9: Backward simulation of a DTMC for a monotone chain

Data: A transition matrix $\mathbf{P}$, in the form of a mapping $\Phi : [0,1] \times \mathcal{E}$
Result: A random state of $\mathcal{E}$
Result: A coupling time τ
begin
  $T \leftarrow 1$
  $\mathtt{u}[T] \leftarrow \mathtt{Unif}([0,1])$
  repeat
    $T \leftarrow 2T$
    $\mathtt{termState}^+ \leftarrow \sigma^+$
    $\mathtt{termState}^- \leftarrow \sigma^-$
    for $t = T$ **downto** $T/2 + 1$ **do**
      $\mathtt{u}[t] \leftarrow \mathtt{Unif}([0,1])$
      $\mathtt{termState}^+ \leftarrow \mathtt{termState}[\Phi(\mathtt{u}[t], \mathtt{termState}^+)]$
      $\mathtt{termState}^- \leftarrow \mathtt{termState}[\Phi(\mathtt{u}[t], \mathtt{termState}^-)]$
    for $t = T/2$ **downto** 1 **do**
      // the number $\mathtt{u}[t]$ has already been computed
      $\mathtt{termState}^+ \leftarrow \mathtt{termState}[\Phi(\mathtt{u}[t], \mathtt{termState}^+)]$
      $\mathtt{termState}^- \leftarrow \mathtt{termState}[\Phi(\mathtt{u}[t], \mathtt{termState}^-)]$
  until *termState*$+$ $=$ *termState*$-$;
return *termState*$^+$, τ

the choice of the next value of σ_k is done by comparing U to p_k. This is equivalent to using the mapping function

$$\Phi_{\boldsymbol{\sigma}}(u, \sigma_k) \;=\; \begin{cases} +1 & \text{if } u \le p_k \\ -1 & \text{if } u > p_k. \end{cases}$$

This function does not depend on the value of its argument σ_k because transitions of this particular model do not depend on the state of the particle. It does depend on the current state $\boldsymbol{\sigma}$ through the value of p_k.

Therefore, since $p_k \le p'_k$, we have, for all u: $\Phi_{\boldsymbol{\sigma}}(u, \sigma_k) \le \Phi_{\boldsymbol{\sigma}'}(u, \sigma'_k)$. The monotonicity condition of the transition mapping is therefore satisfied, and the exact sampling method can be applied to this system.

3.4.3 The Hastings-Metropolis Algorithm

The method of Hastings-Metropolis is a Monte-Carlo Markov Chain algorithm which aims at obtaining samples of a given distribution, through the simulation of a Markov Chain.

The data of the problem is a probability distribution function f on the state space $\mathcal{E}$, and one wants to draw samples of a random variable X with distribution f, that is: for all $i \in \mathcal{E}$,

$$\mathbb{P}(X = i) \;=\; f(i)\ .$$

The practical assumption is that the state space is too large to use one of the methods described in sections B.2.2 or B.2.3.

For every state x, one chooses a probability distribution on $\mathcal{E}$, $q(y|x)$. These distributions are called "instrumental". Then, one defines the function:

$$\rho(x,y) \;=\; \min\left\{\frac{f(y)}{f(x)}\,\frac{q(x|y)}{q(y|x)}, 1\right\}\;. \tag{3.5}$$

The values of this function are nonnegative and less than 1: it can therefore be interpreted as a probability.

The sampling algorithm is then described as Algorithm 10. It uses three random generators: one `InitialSample()` which returns some state in $\mathcal{E}$, the `DiscreteSample` which we have already seen, as well as the usual `Unif`$([0,1])$. It also uses a number of steps N as in Algorithm 3. Indeed, this algorithm simulates, by the forward method, the evolution of a DTMC with transition probabilities:

$$p_{xy} \;=\; \begin{cases} q(y|x)\;\rho(x,y) & \text{if } x \neq y \\ q(x|x) \;+\; \sum_{y\neq x} q(y|x)\;(1-\rho(x,y)) & \text{if } x = y \end{cases} \tag{3.6}$$

(observe that $\rho(x,x) = 1$). It can be checked that the distribution $\pi(\cdot)$ is stationary for this DTMC, whatever the choice of the instrumental function.

Algorithm 10: The Hastings-Metropolis algorithm

```
Data: A distribution function f(·) and a family of instrumental distributions q(·|x)
Data: A time horizon T ∈ ℕ
Result: A sample of the distribution f(·)
begin
    x ← InitialSample()
    for n from 1 to T do
        y = DiscreteSample(q(·|x))
        r ← ρ(x, y) // using (3.5)
        u ← Unif([0, 1])
        if u < r then
            x ← y // keep sample y
        else
            x ← x // reject sample y, keep x
return x
```

For the algorithm to work, some practical assumptions must be satisfied by the family of instrumental distributions:

- it must be algorithmically easy to draw samples from the distribution $q(\cdot|x)$ (small requirement in memory, small computational time);
- the function $\rho(x,y)$ must be easy to calculate;
- the resulting Markov chain must be ergodic.

Among the interesting features of this method, remark that:

- it is not necessary to know how to calculate exactly each $\pi(x)$, but only the ratios $\pi(x)/\pi(y)$. In some situations, this is actually essential. For instance, consider the distribution related to Ising's model described in Sections 3.3.1 and 3.4.2:

$$\pi(\boldsymbol{\sigma}) \;=\; \frac{1}{G}\, e^{-\beta E(\boldsymbol{\sigma})}$$

  where the energy function is given in (3.1), and G is a normalizing constant such that the function $\pi(\cdot)$ is a probability distribution. It is quite possible that the value of G be impossible to calculate exactly. On the other hand,

$$\frac{\pi(\boldsymbol{\sigma})}{\pi(\boldsymbol{\sigma'})} \;=\; e^{-\beta(E(\boldsymbol{\sigma})-E(\boldsymbol{\sigma'}))}$$

  can be computed without G.

- Likewise, it is sufficient to know how to compute the ratio $q(y|x)/q(x|y)$ but it is necessary to know how to sample each distribution $q(\cdot|x)$.

Since the choice of the instrumental distributions is open, one should speak of a "meta"-algorithm. We now discuss two simple choices for instrumental distributions

Independent sampling. The easiest thing to do is to choose one single instrumental distribution $g(\cdot)$ and set: $q(y|x) = g(y)$ for all x in $\mathcal{E}$. In that case, the expression for $\rho(x, y)$ is simply:

$$\rho(x,y) \;=\; \min\left\{\frac{f(y)}{f(x)}\,\frac{g(y)}{g(x)}, 1\right\} \;.$$

Again, the distribution $g(\cdot)$ must be relatively easy to sample from. Often, uniform sampling over the state space $\mathcal{E}$ is possible, in which case $g(x) = g(y)$ for all x and y. The expression of ρ further reduces to:

$$\rho(x,y) \;=\; \min\left\{\frac{f(y)}{f(x)}, 1\right\} \;.$$

With such a choice of the instrumental distribution, Algorithm 10 will accept a new sample y with probability 1 if it increases the relative likelihood $f(y)/f(x)$.

Assuming that several candidate distributions $g(\cdot)$ exist, which one should be chosen? Generally, the algorithm is more efficient if new samples y are accepted more often. In other words, the function $\rho(x, y)$ should be as large as possible. The ideal choice is, of course, to take $g = f$: in that case, $\rho(x, y) = 1$ for all x and y, and samples are always accepted. The algorithm then produces exact samples of the desired distribution in one iteration.

Random walk sampling. When the state space $\mathcal{E}$ is equipped with a neighborhood relation, then it is possible to construct instrumental distributions so that Algorithm 10 realizes a random walk over the state space.

For instance, assume that $\mathcal{E} = \{0, 1, \ldots, n, n+1, \ldots\}$. There is a natural neighborhood relation between consecutive numbers. Accordingly, one may choose, for all $n \geq 1$:

$$q(n+1|n) = q(n-1|n) = 1/2 \;.$$

When $n = 0$, these transitions are modified as: $q(1|0) = q(0|0) = 1/2$. This procedure can be generalized to state spaces with several dimensions.

The big advantage of this choice is that sampling the instrumental distribution is very easy, and takes almost no time or space. The disadvantage is that the progress of the simulation is very local, and that the simulation algorithm may have to run for a long time in order to obtain samples from all parts of the state space. This problem can be made less acute by adding random transitions between distant parts of the state space.

3.4.4 Uniform sampling

3.4.4.1 General Principle

Very often, the practical question is to draw a sample of the uniform distribution on a finite but large space $\mathcal{E}$. This is typically the case in computer science, discrete mathematics or statistical physics: the set $\mathcal{E}$ is then a set of combinatorial objects (words, trees, graphs, arrangements of particles).

In order to do this, one may construct a Markov chain on the state space $\mathcal{E}$, with a transition matrix $\mathbf{P}$ such that the uniform distribution is the unique stationary distribution of $\mathbf{P}$. Denote with M the cardinal of $|\mathcal{E}|$. The stationary distribution on $\mathcal{E}$ is represented by the vector $\pi = M^{-1}\mathbf{1}'$ (remember that $\mathbf{1}$ is the vector of ones, and that "$'$" denotes the transposition of matrices or vectors). Requiring that this vector be stationary for $\mathbf{P}$ means that $M^{-1}\mathbf{1}'\mathbf{P} = M^{-1}\mathbf{1}'$ or equivalently:

$$\mathbf{1}'\mathbf{P} \;=\; \mathbf{1}' \;. \tag{3.7}$$

Matrices $\mathbf{P}$ which satisfy this identity have the property that their columns sum up to 1, as well as their lines. For this reason, they are called *bistochastic* matrices.

The following general construction provides such a matrix. Consider a set $\mathcal{S} = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ of *bijections* (or one-to-one mappings) of $\mathcal{E}$. Among all possible bijections of $\mathcal{E}$, we shall in particular consider *involutions*: mappings σ of $\mathcal{E}$ such that $\sigma \circ \sigma = Id$. Let $(\alpha_1, \ldots, \alpha_k)$ be a probability distribution over $\mathcal{S}$. For each state $i \in \mathcal{E}$, let $j_p = \sigma_p(i)$ for $p = 1..k$. The transition probabilities are then set to:

$$P_{i,j_p} \;=\; \alpha_k \;, \qquad p = 1, \ldots, k \;, \tag{3.8}$$

and 0 otherwise. The convention is that these probabilities add up whenever several of the states j_p are the same.

Lemma 3.5. *For every set of bijections $\mathcal{S}$ and every distribution $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_k)$, the matrix $\mathbf{P}$ defined in* (3.8) *is bistochastic.*

Proof. The sum of the transition probabilities out of i is $\sum_k \alpha_k = 1$ (with the convention above). The matrix is therefore stochastic.

Consider any state i. Since all elements σ_p of $\mathcal{S}$ are bijections, they are invertible, and for all i, there exist exactly k states $j_p = \sigma_p^{-1}(i)$, $p = 1..k$, with a transition $j_p \to i$. The sum of these probabilities is $\sum_k \alpha_k = 1$. The matrix is therefore bistochastic. □

Observe that when all elements of $\mathcal{S}$ are involutions, the matrix $\mathbf{P}$ is actually symmetric: if $P_{i,j} \neq 0$, it means that $j = \sigma_p(i)$ for one or several involutions of $\mathcal{S}$. Then, $\sigma_p(j) = i$, so that transition $i \to j$ is matched by transition $j \to i$ with the same probability.

We further need that the matrix $\mathbf{P}$ be irreducible and aperiodic. These properties have to be checked for the selected set of involutions. Observe that aperiodicity is guaranteed as soon as at least one involution σ_p leaves at least one state i invariant: $\sigma_p(i) = i$.

Another desirable requirement is that the set of involutions $\mathcal{S}$ be rich enough so that the number of transitions between two distinct states is never too large. This is to make simulation algorithms converge faster. On the other hand, this set $\mathcal{S}$ should not be too large so that sampling from it is still fast.

Once the set of involutions and the probability distribution have been selected, a forward simulation algorithm can be used to sample (approximately) from the uniform distribution. This leads to Algorithm 11. The distribution $\boldsymbol{\alpha}$ will usually be the uniform distribution on the set $\mathcal{S}$, but this is not necessarily the best choice. As for other forward simulation algorithms, there is no general rule for setting the time horizon N, unless something is known on the spectral gap of the matrix $\mathbf{P}$ (see section 3.4.1).

In order to use backwards simulation in this context, it is necessary to find a partial order on $\mathcal{E}$ which is preserved by *all* involutions of $\mathcal{S}$: for all $i, j \in \mathcal{E}$ and $\sigma \in \mathcal{S}$, $i \prec j \iff \sigma(i) \prec \sigma(j)$.

Algorithm 11: Uniform sampling using a Markov Chain

Data: A set $\mathcal{S}$ of bijections of $\mathcal{E}$, and a distribution $\boldsymbol{\alpha}$ on the set $\mathcal{S}$, satisfying the requirements of Lemma 3.5
Data: A time horizon $N \in \mathbb{N}$
Result: A sample of the uniform distribution on $\mathcal{E}$
begin
  $x \leftarrow$ `InitialSample()`
  for n **from** *1* **to** N **do**
    $p =$ `DiscreteSample`$(\boldsymbol{\alpha})$
    $x = \sigma_p(x)$
return x

3.4.4.2 Sampling uniformly from a subset

In some situations, the required sample has to be drawn uniformly from a subset $\mathcal{F} \subset \mathcal{E}$. The "natural" involutions acting on objects of the set $\mathcal{E}$ do not necessarily preserve the subset $\mathcal{F}$. In that case, one may use Algorithm 12, which can be seen as a combination of Algorithm 11 and the rejection method of Algorithm 20.

The code of the algorithm assumes that there exist a procedure, `belongsToF(`x`)`, which returns true if the object x belongs to $\mathcal{F}$ and false otherwise.

This algorithm simulates the evolution of a Markov chain on the set $\mathcal{F}$. The uniform distribution on $\mathcal{F}$ is a stationnary distribution, as stated in Lemma 3.6 below. In general, there is no guarantee *a priori* that the Markov chain will be irreducible: this has to be checked for each particular application. On the other hand, the rejection step guarantees that the chain is aperiodic.

Lemma 3.6. *Algorithm 12 simulates a Markov chain on $\mathcal{F}$ with a matrix $\mathbf{P}_{\mathcal{F}}$ which is bistochastic, and therefore admits the uniform distribution as stationary distribution.*

Algorithm 12: Uniform sampling on a subset, using a Markov Chain

Data: A set $\mathcal{S}$ of bijections of $\mathcal{E}$, and a distribution $\boldsymbol{\alpha}$ on the set $\mathcal{S}$, satisfying the requirements of Lemma 3.5
Data: A time horizon $N \in \mathbb{N}$
Data: A subset $\mathcal{F}$, defined by a test procedure `belongsToF()`
Data: A time horizon $N \in \mathbb{N}$
Result: A sample of the uniform distribution on $\mathcal{F}$
begin
  $x \leftarrow$ `InitialSample()`
  for n **from** *1* **to** N **do**
    $p =$ `DiscreteSample`$(\boldsymbol{\alpha})$
    $y = \sigma_p(x)$
    if *belongsToF(y)* **then**
      $x \leftarrow y$ // keep sample y
    else
      $x \leftarrow x$ // reject sample y, keep x
return x

Proof. Consider the set $\mathcal{S}_{\mathcal{F}} = \{\sigma_{1,\mathcal{F}}, \ldots, \sigma_{k,\mathcal{F}}\}$ of functions on $\mathcal{F}$, defined as, for $p = 1, \ldots, k$:

$$\sigma_{p,\mathcal{F}}(i) \;=\; \begin{cases} \sigma_p(i) & \text{if } \sigma_p(i) \in \mathcal{F} \\ i & \text{if } \sigma_p(i) \notin \mathcal{F}. \end{cases}$$

The transitions realized by Algorithm 12 are precisely those resulting from the application of $\sigma_{p,\mathcal{F}}$ on the current state x. It is easily checked that the transformations $\sigma_{p,\mathcal{F}}$ are bijections on $\mathcal{F}$ (in addition, if σ_p is an involution, so is $\sigma_{p,\mathcal{F}}$). Therefore, Lemma 3.5 applies to the set $\mathcal{S}_{\mathcal{F}}$, and the resulting matrix $\mathbf{P}_{\mathcal{F}}$ is bistochastic. □

This result can also be seen as an application of the truncation principle of Theorem 2.2 in a discrete version (not developped in this document). The Markov chain on $\mathcal{E}$ given by the matrix $\mathbf{P}$ is indeed reversible.

3.4.4.3 Illustrations

We now provide some examples of applications of the previous simulation principle. Some of theme have just an illustrative value: there exist better algorithms for sampling uniformly is these sets. In other cases, there are no better alternatives known.

Permutations. Let $\mathcal{E}$ be the set of permutations on N values. There are $N!$ such permutations. Possible choices for the elements of $\mathcal{S}$ are: *transpositions* which exchange two elements (there are $N(N-1)/2$ of them), *circular permutations* which shift all elements ℓ positions to the left or right (there are $N-1$ of them) or the reversal, which consists in exchanging simultaneously elements 1 and N, 2 and $N-1$ etc.

It is known that any permutation can be transformed in any other one by a succession of at most N transpositions. Computing the action of a transposition requires few steps (in contrast with circular permutations or reversal, which require $O(N)$ steps). In addition, transpositions are involutions. Choosing $\mathcal{S}$ as the set of all transpositions is therefore a good choice *a priori*. Sampling uniformly from this set can be performed by choosing uniformly at random the two elements to swap, with a rejection step in case these happen to be the same (or no rejection, which is equivalent to including the identity permutation in the set $\mathcal{S}$).

Sampling from a subset of permutations can be done with Algorithm 12. For instance to sample permutations without fixed points, or some other specific property.

Graphs. In experimental algorithmics, it is often necessary to generate test cases at random to test the efficiency of some algorithm. In [16, 17], the authors have studied the problem of generating directed acyclic graphs (DAGs) uniformly, using Markov chains. DAGs are directed graphs which contain no cycles. We describe here the elements of their technique.

There are several natural transformations that generally apply to directed graphs: *complementing* and edge $i \to j$, that is, deleting it if it exists, creating it if it does not; *reversing* an existing edge. The transformation illustrated in Figure 3.6 applies to triples of nodes $i \to j \to k$ such that no edge exist between i and k. All the transformations listed there are involutions. They conserve the number of edges except the complementing operation.

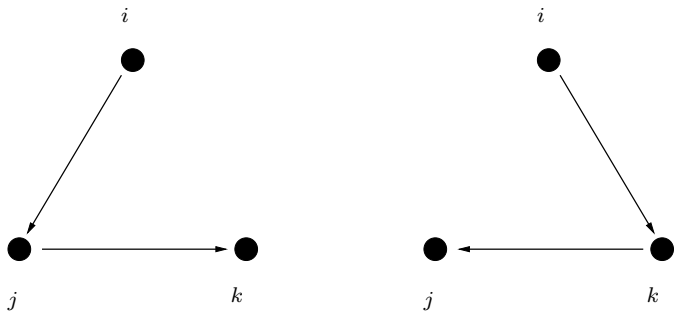


Figure 3.6: Swapping the edges of a triangle

Choosing as $\mathcal{S}$ the set of all edge complementations and reversals fulfills the requirements of irreducibility of the matrix $\mathbf{P}$: it is possible to obtain any directed graph from any other through a sequence of edge complementations: simply delete all edges of the first graph, then add the edges of the second one. It also fulfills the requirement of aperiodicity: in addition to cycles of length 2 generated by involutions, there are cycles of length 3 generated by sequences (reversal, deletion, creation). Algorithm 11 can then be used with this set $\mathcal{S}$ to generate directed graphs uniformly at random, but there are more direct ways to do that.

Assume now that one needs directed *acyclic* graphs. Since the transformations listed above do not preserve the character of being acyclic, one must use Algorithm 12 instead of Algorithm 11. The requirement of irreducibility of the chain truncated to the subset $\mathcal{F} = \{$ directed acyclic graphs$\}$ is satisfied: the argument used above still applies: after deletion of an edge, a DAG is still a DAG. When creating some DAG edge by edge, all intermediate graphs are DAGs also. The requirement of aperiodicity is satisfied as a consequence of rejections, as observed above.

Finally, assume that that one needs *simply connected* DAGs, that is, DAGs such that there exist a sequence of edges connecting each pair of nodes, neglecting orientation. In that case, Algorithm 12 can still be used with the chain truncated to the subset $\mathcal{F} = \{$ simply connected directed acyclic graphs$\}$. Here, it is not immediate to see that the requirement of irreducibility is satisfied. The proof of this fact is provided in [17]. The transformation depicted in Figure 3.6 preserves both acyclicity and connectedness. It may therefore be helpful to speed up simulations, since it is equivalent to several more elementary transformations: complementations and reversals.