Notes on Monte Carlo Markov Chain rules and Simulated Annealing

Carlo Baldassi
(Dated: 17/09/20)

BUILDING A MONTE CARLO MARKOV CHAIN FROM A TARGET DISTRIBUTION

The global balance condition for MCMCs

Consider a finite discrete set $\mathcal{X}$ of size $|\mathcal{X}| = M$. For example, this may be the set $\{1, 2, 3, 4\}$. But it may also be a huge set like the set of all possible Hamiltonian paths of a given undirected complete graph $\mathcal{G} = (V, E)$, in which case $M = |V|!$, or all possible binary strings of some given length n, in which case $M = 2^n$. Since it's discrete and finite, we can choose arbitrarily an indexing of the elements into integers between 1 and M. In the following, we will denote these indices with i or j. So when we write "configuration i" this is a shorthand for "the element of X indexed by the integer i".[1]

Suppose now that we are given a probability distribution over this set, that we denote with ρ. We can encode this in a vector $\vec{\rho} = (\rho_1, \ldots, \rho_M) = (\rho_i)_{i=1}^M$. Obviously, we have the properties that $\forall i : 0 \leq \rho_i \leq 1$ and $\sum_{i=1}^M \rho_i = 1$.

Now consider a Markov process in which for each starting configuration i we have a probability distribution that tells us what is the probability of ending up in a configuration j. We can write this as $P(i \to j)$. Notice that this is a conditional probability (it's conditioned over i), so that sometimes you may see it written as $P(j|i)$. One obvious property is that $\forall i, j : 0 \leq P(i \to j) \leq 1$. We also have a normalization property: starting from i, we must end somewhere in $\mathcal{X}$, thus $P(i \to \mathcal{X}) = \sum_{j=1}^M P(i \to j) = 1$.

Because it's nice to be able to use linear algebra tools, we now change notation slightly and encode these transition probabilities in a matrix Q, by simply writing $Q_{ji} \equiv P(i \to j)$. Notice the order of the indices in the matrix though! Our previous normalization condition becomes

$$\forall i : \quad \sum_{k=1}^M Q_{ki} = 1 \tag{1}$$

Now we can ask: what is the probability of getting to configuration j, assuming that we extract a starting configuration according to $\vec{\rho}$ and we perform one step of the Markov process Q? The answer to this is simply the sum over all possible starting points i of the probability of starting at i times the probability of going from i to j, that is:

$$\sum_{i=1}^M \rho_i P(i \to j) \equiv \sum_{i=1}^M Q_{ji} \rho_i = (Q\vec{\rho})_j$$

The last step is just a rewriting using matrix notation, and it means that we can encode the probability distribution of the outcome in a vector which is just the product of the matrix Q and the vector $\vec{\rho}$, i.e. $Q\vec{\rho}$. The j-th component of this vector is the probability of landing at configuration j. If we iterate the Markov process, we thus clearly just need to multiply by the transition matrix Q on the left again, so that we get $Q^2\vec{\rho}$, then $Q^3\vec{\rho}$, etc. In general, after the n-th step we get $Q^n\vec{\rho}$.

Now it's clear that, if we have Q such that $Q\vec{\rho} = \vec{\rho}$ then $Q^n\vec{\rho} = \vec{\rho}$ for all n and thus our process is so-called *stationary*.

So the question is: given $\vec{\rho}$, can we find a matrix Q such that the resulting Markov process is stationary?

$$\text{given } \vec{\rho}, \text{ find } Q \text{ such that: } Q\vec{\rho} = \vec{\rho} \tag{2}$$

Let's rewrite this condition. Using the normalization condition (1) and isolating the diagonal terms in the matrix

[1] This notation is convenient here, but keep in mind that it's rather unusual in applications, where i may represent for example a node in a graph, or something like that, while the configuration is usually written more explicitly in a form that makes sense for the particular problem that one is studying.

Q:

$$\forall i: \quad \sum_{j=1}^{M} Q_{ij}\rho_j = \rho_i$$

$$\sum_{j\neq i} Q_{ij}\rho_j + Q_{ii}\rho_i = \left(\sum_{k=1}^{M} Q_{ki}\right)\rho_i$$

$$\sum_{j\neq i} Q_{ij}\rho_j + Q_{ii}\rho_i = \sum_{k\neq i} Q_{ki}\rho_i + Q_{ii}\rho_i$$

We arrive at this expression:

$$\forall i: \quad \sum_{k\neq i} Q_{ki}\rho_i = \sum_{j\neq i} Q_{ij}\rho_j \tag{3}$$

This formula, which is equivalent to the stationarity condition, is known as the "global balance" condition. It expresses the condition that at each step the total "outflow" of probabilities for a configuration i equals the total "influx" of probabilities for that configuration.

Detailed balance and the proposal-accept/reject scheme

The problem now is that the expression (3) has too many solutions and does not allow us to choose any particular form for the matrix Q that we are seeking. Therefore, we proceed as follows: we impose a more restrictive (but simpler to analyze) condition, such that if we can satisfy that then we automatically get a solution to (3).

This new more restrictive condition is called "detailed balance", and it reads:

$$\forall i, j: \quad Q_{ij}\rho_j = Q_{ji}\rho_i \tag{4}$$

It's easy to see that if this holds then global balance is also satisfied: just keep i fixed and sum over $j \neq i$. The interpretation is that the probability flows are now balanced for every pair of configurations.

Next, we envision a process in which the transitions happen in the following way: whenever we are at some configuration i, we first propose a new configuration j, different from i, and then we decide whether to accept the proposal or not. If we reject the proposal then we just remain at i.

To realize this idea mathematically, we split the off-diagonal transition probabilities Q_{ij} into a "proposal" part C_{ij} and an "acceptance" part A_{ij}:

$$Q_{ij} = C_{ij}A_{ij} \qquad \text{if } i \neq j \tag{5}$$

In order for this to make sense, the proposal matrix should be normalized along columns, just like the Q matrix, i.e. $\forall i: \sum_j C_{ji} = 1$, again because we must propose something. It should also be 0 on the diagonal, $C_{ii} = 0$ for all i. This means that, starting from some state i, we always propose some *other* state j, thus it ensures that for any i there is always at least some non-zero value of C_{ij} with $j \neq i$. Notice that this also means that the diagonal entries in matrix A are irrelevant and we might set them so 0 as well.

The diagonal case for the matrix Q is simply defined instead as the rejection rate: the probability of not accepting the proposed move. This is fixed by normalization, eq. (1), as $Q_{ii} = 1 - \sum_{k\neq i} Q_{ki}$. In the following, we will focus on the A and C matrices and thus always implicitly assume $i \neq j$.

Symmetric proposals, exponential reparametrization, acceptance rules

Let us now assume symmetric proposals (the more general case is treated at the end of this note):

$$C_{ij} = C_{ji} \tag{6}$$

We also define the quantities u_i, Δ_{ij} by:

$$\rho_i = \frac{1}{Z}e^{u_i} \tag{7}$$

$$\Delta_{ij} = u_i - u_j \tag{8}$$

The first expression is just a remapping of $\vec{\rho}$, basically a change of variables.[2] The Z in the first equation is a normalization constant equal to $\sum_k e^{u_k}$. One of our goals will be to try to avoid having to ever compute Z explicitly: it's a sum over all the configurations in X, i.e. it's a sum over M elements, and as we said M may be huge, too much even for a computer to handle in reasonable times.

Note that there are no domain restrictions on u_i, it can take values in all $\mathbb{R}$. Also, this remapping is well defined (i.e. unambiguous) except for an irrelevant additive constant that doesn't depend on i.[3] Also note that $\Delta_{ij} = -\Delta_{ji}$.

We will now also restrict ourselves to acceptance rules of a particular form, in which the element A_{ij} is a function of only Δ_{ij}. (Remember that at this stage we are still trying to find whatever solution for our stationarity condition, so as long as we end up with a valid Q we can keep adding restrictions.) We express it through an auxiliary function $g(x)$, defined by:

$$\ln A_{ij} = \frac{1}{2}\left(\Delta_{ij} + g\left(\Delta_{ij}\right)\right) \tag{9}$$

Since A_{ij} must be a probability, and thus less than 1, the logarithm must be negative and we obtain this condition on g:

$$\forall x: \quad g(x) \leq -x \tag{10}$$

With these definitions, the detailed balance condition eq. (4) becomes:

$$\forall i,j: \quad e^{\frac{1}{2}(\Delta_{ij}+g(\Delta_{ij}))}e^{u_j} = e^{\frac{1}{2}(-\Delta_{ij}+g(-\Delta_{ij}))}e^{u_i}$$

which after simple algebraic manipulations simplifies to the requirement:

$$\forall x: \quad g(x) = g(-x) \tag{11}$$

This also implies that the bound of eq. (10) becomes stricter:

$$\forall x: \quad g(x) \leq -|x| \tag{12}$$

We have finally arrived at the following result: as long as we can find any function $g(x)$ that is symmetric (criterion (11)) and bounded as in (12), we can then use it to build a matrix A_{ij} using eq. (9). Then we can pick (almost arbitrarily[4]) a symmetric choice matrix C and finally obtain a matrix Q that satisfies detailed balance, eq. (4), and thus our stationarity condition, eq. (3), and therefore solve our original problem (2).

We still have to choose g. Here are two common choices (the first one being by far the most common):

1. Metropolis rule: $g(x) = -|x|$. This is obviously the simplest one. Then

$$\begin{aligned} A_{ij} = e^{\frac{1}{2}(\Delta_{ij}-|\Delta_{ij}|)} &= \begin{cases} 1 & \text{if } \Delta_{ij} \geq 0 \\ e^{\Delta_{ij}} & \text{if } \Delta_{ij} < 0 \end{cases} \\ &= \min\left(1, e^{\Delta_{ij}}\right) = \min\left(1, \frac{\rho_i}{\rho_j}\right) \end{aligned} \tag{13}$$

2. Glauber (AKA Berker) rule: $g(x) = -2\ln\left(2\cosh\frac{x}{2}\right)$. Then

$$\begin{aligned} A_{ij} &= e^{\frac{1}{2}\left(\Delta_{ij} - 2\ln\left(2\cosh\frac{\Delta_{ij}}{2}\right)\right)} \\ &= \frac{e^{\frac{1}{2}(u_i-u_j)}}{e^{\frac{1}{2}(u_i-u_j)} + e^{-\frac{1}{2}(u_i-u_j)}} \\ &= \frac{e^{u_i}}{e^{u_i}+e^{u_j}} = \frac{\rho_i}{\rho_i+\rho_j} \end{aligned} \tag{14}$$

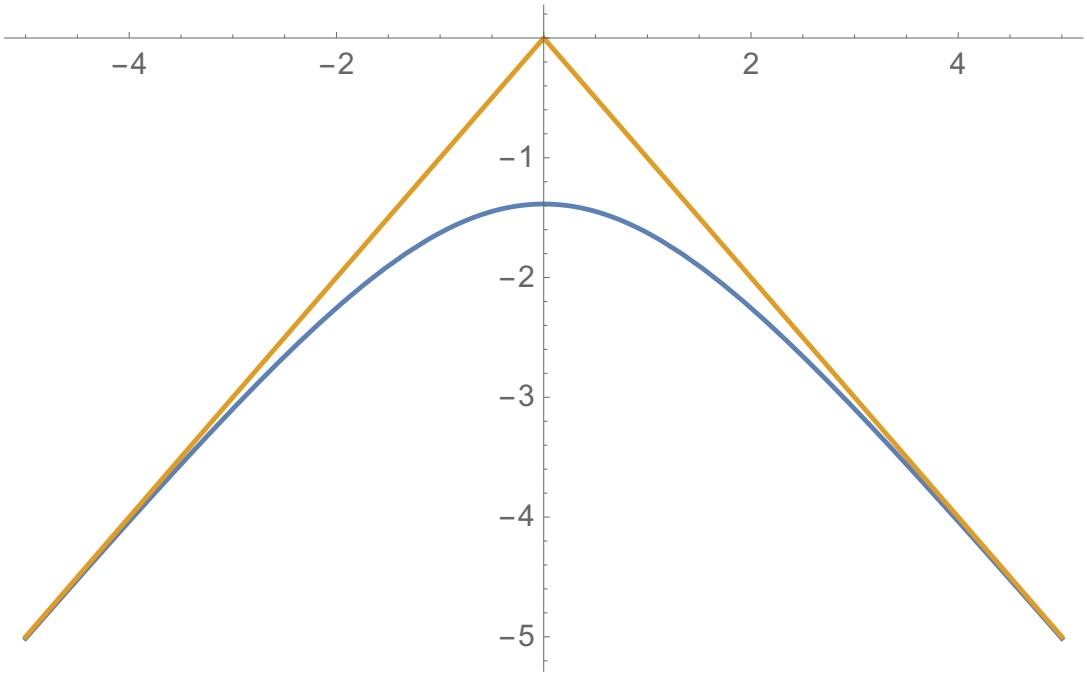
---

[2] Mathematically, this doesn't work for those i where $\rho_i = 0$, but this is just a minor annoyance and not a substantial problem: just expunge those configurations from X and start over. Thus assume that $0 < \rho_i \leq 1$.

[3] Here's one way to find a valid vector $\vec{u}$: set $u_1 = 0$, then for all $i \geq 2$ set $u_i = u_1 + \ln(\rho_i/\rho_1)$. It may seem surprising, but you can check that it works by substituting it back into eq. (7).
Suppose now that you have found a valid $\vec{u}$ and consider an alternative $\vec{u}'$ with $u'_i = u_i + c$, i.e. shift the whole vector $\vec{u}$ by a constant quantity. This produces the same probability vector $\vec{\rho}$, since $\frac{e^{u'_i}}{\sum_k e^{u'_k}} = \frac{e^{u_i+c}}{\sum_k e^{u_k+c}} = \frac{e^c e^{u_i}}{e^c \sum_k e^{u_k}} = \frac{e^{u_i}}{\sum_k e^{u_k}} = \rho_i$. Furthermore, $\Delta_{ij} = u_i - u_j = u'_i - u'_j$ is also unaffected by the shift. Therefore we say that eq. (7) defines $\vec{u}$ uniquely up to an additive constant.

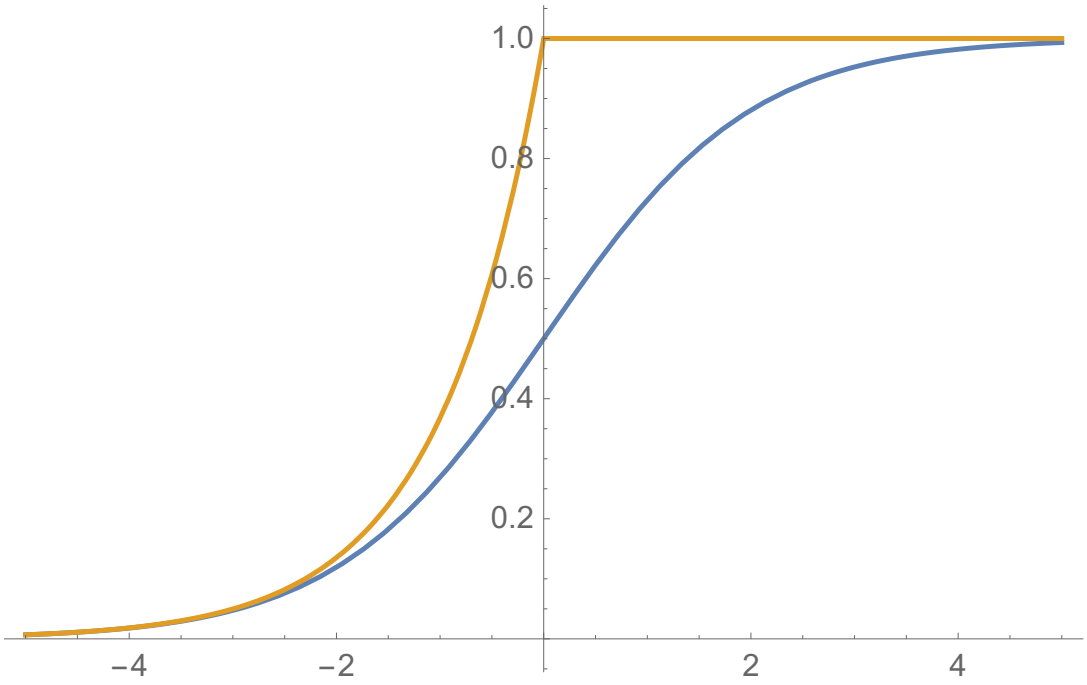
[4] Apart from the requirements of zero diagonal and normalization along columns, mentioned above, we generally impose some additional mild conditions on C. This is because we normally want to ensure that the resulting matrix Q has $\vec{\rho}$ as its *only* stationary state, because we want to ensure that, for *any* initial vector of probabilities $\vec{v}$, the Markov chain process converges to our desired stationary limit, $\lim_{n\to\infty} Q^n \vec{v} = \rho$. Mathematically, this can be ensured by requiring that C has only one eigenvector with eigenvalue 1. One important condition that we need to ensure in practice is that for any two configurations i and j there is a finite sequence of moves k_1, k_2, ..., k_n such that the probability of the sequence is non-zero, i.e. $C_{jk_n} \ldots C_{k_2k_1} C_{k_1 i} > 0$.

Here is a plot comparing the g function for Metropolis (orange) and Glauber (blue). The fact that Metropolis saturates the upper bound on g, eq. (12), provides a very intuitive explanation for the circumstance that the Metropolis rule is generally preferable, since it will have the highest possible acceptance rate compatible with the bounds and the detailed balance condition (in the "symmetric proposals" scenario).



(It is possible to interpolate between the two expressions with $g(x) = -\alpha^{-1} \ln(2\cosh(\alpha x))$ where $\alpha = 1/2$ is the Glauber case and $\alpha \to \infty$ the Metropolis case. In general a lower α means lower curves and thus lower acceptance rates. This is just a mildly interesting observation of no real practical use though, as far as I can tell.)

Comparing the resulting acceptance rates A_{ij} as a function of Δ_{ij} for Metropolis and Glauber we get this:



This shows directly that the Metropolis acceptance rate for any Δ_{ij} is higher, as we expected.

Introducing an "inverse temperature" parameter β: the Gibbs distribution

Now suppose that our target distribution is defined using some "cost" function $E : \mathcal{X} \to \mathbb{R}$ and an arbitrary positive parameter β, like this:

$$\rho_i = \frac{e^{-\beta E(i)}}{Z(\beta)} \tag{15}$$

$$Z(\beta) = \sum_k e^{-\beta E(k)} \tag{16}$$

This form originates in Physics studies and is known as the Boltzmann distribution, or Gibbs distribution. The extra parameter β is non-negative and it's called an "inverse temperature", because that is its meaning when this formula is used to describe physical systems. Also, in that context, $E(i)$ is interpreted as an energy function, and $Z(\beta)$ is called the "partition function". The usefulness of this distribution is not at all restricted to Physics though, and β doesn't need to have anything to do with an actual temperature.

With our notation above, we see that we can simply set $u_i = -\beta E(i)$ and then derive the Metropolis rule as $A_{ij} = \min\left(1, e^{-\beta(E(i)-E(j))}\right)$. Recall that in this notation j is the starting configuration and i the proposed move, thus $\Delta E_{ij} = E(i) - E(j)$ is the comparative "cost" of accepting the move. If it is negative than the move is convenient and it is always accepted, if it is positive than it is only accepted with some probability that depends on β and rapidly becomes lower as β increases, going to 0 as $\beta \to \infty$.

SIMULATED ANNEALING

Attacking an optimization problem by MCMC sampling

Say we have an optimization problem over a discrete set $\mathcal{X}$, in which we want to find the configuration i that produces the minimum value of some cost function $E(i)$. Or at least some "good" value, for which the cost is low. Now consider the expression (15) for some $\beta > 0$. Simply due to the properties of the exponential, the probability ρ_i will be higher when $E(i)$ is lower. Therefore, if we manage to draw samples from X according to the probability distribution $\vec{\rho}$ we will favor configurations of low cost. Using Monte Carlo Markov Chains as described above allows us to do just that without the need to ever compute explicitly the normalization factor $Z(\beta)$, which as we said may be too hard to do computationally. Informally speaking, we just need to start from some arbitrary configuration i_0 and run MCMC a for a T steps, obtaining i_i, i_2, ..., i_T. If we have enough steps (how many depends on the detail of the problem, on our choice of the proposal matrix C, and on β) our final configuration i_T will be (very nearly) distributed according to our desired probability distribution $\vec{\rho}$.

The role of the proposal matrix C

As a matter of computational efficiency, it is also important to point out that at each step of MCMC we will have some current configuration j and a proposed (different) configuration i extracted according to C_{ij}, and we will need to compute $\Delta E_{ij} = E(i) - E(j)$. We would like this computation to be cheap so that we can perform as many MCMC steps as possible. In many cases, this occurs when the configurations i and j are sufficiently "similar" (what this means depends on the details of the problem at hand). For example, say that our space X is made of all possible permutations of the numbers from 0 up to $n-1$. There are $|X|=n!$ such permutations, which grows very fast (super-exponentially) with n. Two "sufficiently similar" permutations i and j might be equal except for two entries, e.g. take $n = 7$ and $i = (\mathbf{3}, 4, 0, \mathbf{1}, 6, 2, 5)$ and $j = (\mathbf{1}, 4, 0, \mathbf{3}, 6, 2, 5)$. It might be that the difference in the costs of these two configurations can be computed in $O(1)$ operations, whereas if we wanted to compute each cost from scratch we would need $O(n)$ operations. In order to ensure that we are always in this situation, we must limit our proposals to "similar" configurations: for any given j, set $C_{ij} = 0$ for all i which are not "similar" to j in this sense. Typically this means that only a tiny fraction of the entries of C are going to be non-zero. As a result our MCMC always moves at most by "small increments" at each step.

Very often, it is also the case that this choice of only proposing "local" moves in which the configuration is not changed too much is also crucial to have reasonably good chances that the proposed move will be accepted: if we propose moves that are mostly unrelated to the current configuration, and the current configuration is, let's say, not horrible in terms of the cost, chances are that most "blind" proposals will just destroy the configuration, while small

increments have a higher chance of doing something good or at least not too disruptive. (More on this rather vague point below.)

The extreme cases $\beta = 0$ and $\beta \to \infty$

The next observation is about what happens for different values of β. At $\beta = 0$, the distribution ρ_i is just uniform. If we run MCMC we are just going to accept all moves, and thus perform a so-called "random walk" in the set X: our configuration moves around aimlessly without caring for our cost function $E(i)$ at all. At the other extreme, when β is very large and tends to infinity, the probability ρ_i becomes very sharply concentrated on the minima of the cost $E(i)$. To see this, call $i^\star$ the configuration with the minimum cost, $i^\star = \arg\min_i E(i)$, and suppose that it is unique. Then write

$$\begin{aligned}
\lim_{\beta\to\infty} \frac{e^{-\beta E(i)}}{\sum_k e^{-\beta E(k)}} &= \lim_{\beta\to\infty} \frac{e^{-\beta E(i)}}{e^{-\beta E(i^\star)} + \sum_{k\neq i^\star} e^{-\beta E(k)}} \\
&= \lim_{\beta\to\infty} \frac{e^{-\beta (E(i) - E(i^\star))}}{1 + \sum_{k\neq i^\star} e^{-\beta (E(k) - E(i^\star))}} \\
&= \begin{cases} 1 & \text{if } i = i^\star \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

(In the second line we have divided both the numerator and the denominator by $e^{-\beta E(i^\star)}$ and in the third line we have performed the limit by exploiting the fact that for all k with $k \neq i^\star$ we have $E(k) \geq E(i^\star)$ by definition of $i^\star$.) It's easy to see with the same argument that it there is more than one optimal configuration with the same minimum cost, the probability would just be uniform over the optima and zero everywhere else.

Therefore, if we were able to sample i from ρ_i at $\beta \to \infty$ we would surely get an optimal configuration $i^\star$! The problem of doing so in practice is that now if we start from a random initial condition i_0 we are going to only accept moves that decrease the cost. Why is that a problem? Because of the following facts, that occur in many situations of practical interest:

- The size of the set $\mathcal{X}$ may be huge and the number of optimal (or even "good enough") configurations comparatively minuscule. Thus "guessing" an initial condition i_0 which is already optimal is just out of the question.
- As we said, the structure of our proposal matrix C is normally such that we will only propose moves that change small pieces of the current configuration: our candidate moves are "local" to the current configuration. The more non-local the moves, the larger the space of candidate moves, the higher the chance that our proposed move gets us to a random configuration that makes no sense.
- It very much possible that there are a huge number of "locally optimal" configurations, in the sense that if we are in one such configuration, call it j, all the configurations i which are reachable from i via C, i.e. all the configurations for which $C_{ij} > 0$, have a higher cost than j. And yet the cost $E(j)$ may still be very large.

Under these conditions, our random initial pick i_0 would likely have a high cost, and after a few moves we would end up in a local minimum with a still-too-high cost and get stuck there indefinitely because we would simply reject each of the proposed moves. Our algorithm would get trapped! How can we try to solve this issue? What we would like to do is start from an i_0 which is already rather good, possibly "close" to the optimum. How can we get such an i_0? By using the result of a previous run of MCMC with a slightly lower β, which is not as good at optimizing but is also less prone to getting stuck in local optima, as we'll discuss in the next paragraph.

Annealing as an optimization scheme

In between the two extremes $\beta = 0$ and $\beta \to \infty$ the situation is intermediate between the two scenarios described above. Our MCMC process would still be more likely to accept moves that decrease the current cost as opposed to those that increase it (this is more and more true for large β). However, it would also have a chance to escape from local minima (this is less and less true for large β though). **The idea of Simulated Annealing is to perform a MCMC on the Gibbs distribution starting at small β and gradually increase it. (The process of increasing β is called "annealing".)** In this way the initial iterations are more "exploratory" since the configurations are almost free to evolve, but as time passes the algorithm becomes more and more "greedy", rejecting more and more those moves

that worsen the current situation. The hope is that this works by "building up" a good configuration incrementally, by starting at random and fixing the configuration a piece at a time. In the initial phases the algorithm is willing to attempt to get worse sometimes, taking the chance that maybe another subsequent change will bring an overall improvement. But at the end, when "most" of the configuration is already in fairly good shape, only "local fixes" are accepted, with ever more rare exceptions.

This scheme as it's defined is clearly very heuristic[5] and many details and considerations are left at an intuitive level. It is actually possible to have a more rigorous treatment and prove (under some conditions) that the algorithm will reach the global optimum with high probability if given enough time and if the annealing process (the gradual increase of β) is sufficiently gradual. This sort of treatment is out of the scope of this note and furthermore is not that useful in practice (unfortunately), since in order to satisfy the mathematical guarantees the resulting algorithm would simply be impractically slow. What is done in practice is the following:

- Given a cost function E, choose a suitable representation for the configurations of the problem
- Choose the kind of moves that we want to propose (i.e. choose how to set the elements C_{ij} keeping in mind that we always need to set the diagonal elements to 0). Possibly, in such a way that computing the cost differences ΔE_{ij} is cheap, ideally $O(1)$.
- Choose how to set the initial configuration (normally, it's just at random)
- Choose an "annealing protocol": A sequence of increasing values of β, and a number of MCMC steps to perform for each value of β.

The first three points are all related, may sometimes pose some serious programming challenges and different choices may lead to radically different results both from the point of view of raw computational efficiency (time per attempted move) and from the point of view of the "exploration efficiency" of the Markov chain (fraction of "useful moves" that get accepted).

A simple and popular annealing protocol consists in just growing β linearly, thus we only need to choose the initial and final values β_0 and β_1 and a number of annealing steps. It is also generally a good idea to just do some steps at $\beta = \infty$ right at the end.[6] Normally it doesn't make much of a difference if we scale up the number of annealing steps by some factor and scale down the number of MCMC steps by the same factor, what counts is mostly the overall number of steps, which is the product of the two.

Some rules of thumb for setting the parameters

The choice of β_0 and β_1 and the number of steps is very much heuristic and depends on the problem, on the representation of the configurations that we have chosen, on the choice of C, etc. It needs to be done by some trial-and-error and with a little bit of experience. Here are some general rules of thumb:

- The initial β_0 should be such that at the beginning the acceptance rate (the fraction of proposed moves that is accepted, observed over a few hundreds of MCMC steps) is fairly high, say somewhere between 30% and 80%
- The acceptance rate should in most cases decrease gradually and reach nearly (but not quite) zero only at the final β_1. A small acceptance rate means that most proposed moves are rejected, i.e. that most CPU time is being wasted, and if that happens too early then probably β_1 is too high.
- The overall number of steps should be set in such a way that the algorithm has enough time to find good configurations (in principle, the more time the better, in the limit of infinite time we would get to the optimum), but we also don't want to spend more time than necessary (the quicker we can find an optimum the better). Our goal is to find "the least number of steps such that the algorithm still works fine".

The virtue of this algorithm is that is very generic, it can be applied to nearly any discrete optimization problem, it often (although not always) produces good or at least reasonable results even for some very hard problems.

---

[5] The origin of the algorithm is in trying to mimic what happens in physical systems (in particular, metals) that get slowly cooled down, starting as amorphous at high temperature and ending up with all atoms arranged in an ordered lattice at low temperatures – hence the name. In the analogy, the cost is the internal energy of the metal, that is lowest in the ordered configuration; the temperature is $T = 1/\beta$ and thus going from high to low temperature means going from low to high β.

[6] An alternative, similar but not equivalent, protocol that is fairly popular is to scale the "temperature" linearly instead: choose an initial value T_0, a number of annealing steps, and go from T_0 down to 0 in equally spaced intervals. For any "temperature" just set $\beta = 1/T$; the last step is thus done at $\beta = \infty$.

EXTRA: GENERALIZATION OF MCMC ACCEPTANCE RULES FOR NON-SYMMETRIC PROPOSAL MATRICES

Let us now roll back a little and generalize the MCMC scheme to the case where the proposal C is not necessarily symmetric. We will still want to satisfy the detailed balance condition eq. (4) though, and thus we observe that $C_{ij} = 0 \iff C_{ji} = 0$, so at least the structure of the zeros of the matrix must be symmetric. In the following, we will only consider the case $C_{ij} > 0$, $C_{ji} > 0$, since otherwise detailed balance is just trivially satisfied regardless of A_{ij} and A_{ji}.

Furthermore we will consider a more general scenario in which we lift the restriction that A_{ij} is a function of only Δ_{ij}. We define g_{ij} by writing:

$$\ln A_{ij} = \frac{1}{2}\left(\Delta_{ij} + g_{ij}\right) \tag{17}$$

which, as before, implies the condition:

$$\forall i, j: \quad g_{ij} \leq -\Delta_{ij} \tag{18}$$

and the detailed balance condition becomes:

$$\forall i, j: \quad C_{ij} e^{\frac{1}{2}(\Delta_{ij}+g_{ij})} e^{u_j} = C_{ji} e^{\frac{1}{2}(-\Delta_{ij}+g_{ji})} e^{u_i}$$

which after simple manipulations simplifies to the requirement:

$$\forall i, j: \quad g_{ij} + 2\ln C_{ij} = g_{ji} + 2\ln C_{ji} \tag{19}$$

Using this together with eq. (18) gives:

$$\begin{aligned} g_{ji} + 2\ln C_{ji} &\leq -\Delta_{ij} + 2\ln C_{ij} \\ g_{ji} &\leq -\Delta_{ij} + 2\ln \frac{C_{ij}}{C_{ji}} \\ g_{ij} &\leq \Delta_{ij} + 2\ln \frac{C_{ji}}{C_{ij}} \end{aligned}$$

where in the last step we just switched the indices i and j and then used $\Delta_{ji} = -\Delta_{ij}$.

Putting this together with eq. (18) again, we get:

$$g_{ij} \leq \min\left(-\Delta_{ij}, \Delta_{ij} + 2\ln \frac{C_{ji}}{C_{ij}}\right) \tag{20}$$

If we want to saturate this bound we need to set

$$g_{ij} = \min\left(-\Delta_{ij}, \Delta_{ij} + 2\ln \frac{C_{ji}}{C_{ij}}\right) \tag{21}$$

which, analogously to the previous case, results in the Metropolis-Hastings rule for the generalized scenario:

$$\begin{aligned} A_{ij} &= e^{\frac{1}{2}\left(\Delta_{ij} + \min\left(-\Delta_{ij}, \Delta_{ij} + 2\ln \frac{C_{ji}}{C_{ij}}\right)\right)} \\ &= \begin{cases} 1 & \text{if } \Delta_{ij} \geq \ln \frac{C_{ij}}{C_{ji}} \\ \frac{C_{ji}}{C_{ij}} e^{\Delta_{ij}} & \text{otherwise} \end{cases} \\ &= \min\left(1, \frac{C_{ji}}{C_{ij}} e^{\Delta_{ij}}\right) = \min\left(1, \frac{C_{ji}\rho_i}{C_{ij}\rho_j}\right) \end{aligned} \tag{22}$$