Simulated Annealing and Global Optimization

Jason Cantarella*

1. OVERVIEW

We've now discussed two methods for the unconstrained optimization optimization problem

$$\text{given } f : \mathbb{R}^n \to \mathbb{R}, \text{ minimize } f(x).$$

Nelder-Mead (when you don't know ∇f) and steepest descent/conjugate gradient (when you do). Both of these methods are based on attempting to generate a sequence of positions x_k with monotonically decreasing $f(x_k)$ in the hopes that the $x_k \to x^*$, the global minimum for f. If f is a convex function (this happens surprisingly often), and has only one local minimum, these methods are exactly the right thing to use: you know in that case that there is only one local min of the function and that it is the global min.

In this case, in the plane, Nelder-Mead is guaranteed to converge to the minimum (nobody has proved it for $\mathbb{R}^n$, even for convex functions, as far as I know). And steepest descent/conjugate gradient is guaranteed to converge under some very reasonable assumptions about rate of decay of $|\nabla f|$ to zero as we approach the min.

However, for a non-convex objective function f the situation is really different. First, we can expect there to be a collection of local minima for the function located in "basins" which are surrounded by "rims". These basins are the domains of attraction for the (negative) gradient *flow*[1] of the function f. If a minimization algorithm is started in the basin corresponding to a local min, it is entirely possible that the sequence of points x_k generated by the algorithm may have to include points with *larger* function values than $f(x_0)$ in order to reach the global min.

The first observation we can make is that local information can't tell us which way to

*University of Georgia, Mathematics Department, Athens GA

[1] These are in principle different from the basins of attraction of, say, the steepest descent algorithm because the finite size steps can skip over dips in the function landscape which radically change the course of the gradient flow.

direct our efforts to find a global min. After all, you can easily imagine a collection of objective functions which are exactly the same in a neightborhood of the origin, but have global minima in very different locations. Therefore, we are going to have to make some choices randomly in our algorithm. To formalize things, suppose we have

- A *state space* Ω which describes the set of inputs to be searched. This space can be $\mathbb{R}^n$, or any subspace of $\mathbb{R}^n$, including finite subsets, bounded regions, and complicated submanifolds.
- An objective function (usually called an energy function) $E\colon \mathbb{R}^n \to \mathbb{R}$.
- A set or space of moves η which join pairs of points in Ω, together with an algorithm which randomly samples the space of moves $\eta(w)$ which join other points to any given $w \in \Omega$.

Example. If we wish to search a graph, then Ω consists of the vertices of the graph, $\eta(w)$ consists of the vertices joined to w by an edge, and a sampling algorithm consists of choosing among the elements of $\eta(w)$ with equal probability.

Example. If we wish to optimize a function over $\mathbb{R}^n$, then $\Omega = \mathbb{R}^n$, the space of moves $\eta(w)$ might be the unit ball, with vector v joining w to $w + v$, and a sampling algorithm consists of choosing a vector at random in the unit ball according to volume.

A simulated annealing algorithm is given by the following procedure. Start with any point x_0 in Ω.

- Given x_k, choose a candidate x_{k+1} from $\eta(x_k)$ using the random sampling algorithm for $\eta(x_k)$ and compute $\Delta(E) = E(x_{k+1}) - E(x_k)$.
- Evaluate an "acceptance rule" $\alpha(\Delta E, k)$ to give the probability of accepting x_{k+1} as the next configuration.
  - With probability $\alpha(\Delta E, k)$, update x_k to x_{k+1} and continue.
  - With probability $1 - \alpha(\Delta E, k)$, return to the previous step and choose another move.

We require the acceptance rule α to obey some reasonable properties at every k.

- If $\Delta E < 0$, then $\alpha(\Delta E, k) = 1$. That is, downhill steps are always accepted.
- If $\Delta E > 0$, then $\alpha(\Delta E, K)$ is monotone decreasing in ΔE. That is, steps which increase energy by a lot are less likely than steps which increase energy a little.
- As $\Delta E \to \infty$, $\alpha(\Delta E, k) \to 0$. That is, huge upward steps in energy are unlikely to be accepted.

This simple framework for minimization turns out to be surprisingly effective, and while the choices of move sets, sampling algorithms, and acceptance rules definitely change the performance of the algorithm, it is rare for them to entirely stall convergence. The basic idea is very simple: the acceptance of occasional upward steps allow you to escape the basin of attraction of a bad local minimum, while the bias in favor of downhill steps leads the algorithm towards minimizers. For small k, we will leave the acceptance probability of upwards relatively high, in order to allow the algorithm to explore the energy landscape. As the algorithm goes on, we will gradually reduce the probability of accepting an uphill step in order to ensure that we spend more time in deeper basins, increasing the chance that we will stay in the basin of the global minimum. In the endgame, we will reduce the probability of accepting an uphill step to zero in order to find the bottom of the current basin.

2. DEMONSTRATION: A SELF-ASSEMBLING SHAPE

The name "annealing" refers to a process for encouraging crystal formation in metals. An orderly crystal structure is a (global) minimizer for a complicated potential energy given by the attractions experienced by the metal atoms. There are certainly plenty of local minima for this potential given by lining up small clusters of atoms together in a way which does not align globally. When the metal is at high temperature, random thermal motions are very energetic, and can shift the atoms to configurations with higher potential energy. As the temperature cools, these random motions contain less energy, and are more likely to shift the atoms into a configuration of lower potential energy. If the cooling is slow enough, extremely large crystals can be formed with millions of atoms beftween defects. A version of this process was practiced by the legendary swordsmiths of medieval Japan.

Demonstration. There's a self-assembling dodecahedron on the shelves in my office behind the desk. It's in a plastic container. The structure is created from 12 identical faces created with a 3d printer. They are joined by magnets, and the potential energy of the

configuration is lowest when the magnets are in close contact. The assembled state is a global minimum of this energy.

Break the thing apart, and put the pieces in the tub and shake it and tumble it. It takes about 5 minutes, but it really will put itself back together. If you get frustrated with the demo, or just don't want to do it, you can show this YouTube link to a similar demonstration.

`http://www.youtube.com/watch?v=X-8MP7g8XOE`

3. THE METROPOLIS ALGORITHM

The original simulated annealing algorithm was proposed in 1953 by Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller, and works by setting giving the acceptance criterion

Definition 1. The Metropolis acceptance criterion is this. Given a parameter T (called the temperature),

$$\alpha(\Delta E, k) = \begin{cases} e^{-\Delta E/T} & \text{if } \Delta E > 0. \\ 1 & \text{if } \Delta E \leq 0. \end{cases}$$

As $k \to \infty$, $T \to 0$. The rate at which one reduces T is called the *cooling schedule* and for now we'll leave it unspecified.

Of course, it's completely mysterious right now *why* this acceptance rule would have been chosen. And, indeed, in order to do so, we'll have to explain some things about statistical mechanics and Markov chains. The basic idea here is this: since the operation of the algorithm is itself random, we can't easily predict the outcome of a single run. It's possible, for instance, that if any adjacent state to the start point has higher energy, then the algorithm will *never* succeed at picking x_1. However, this is very unlikely.

What we want to do instead is imagine that we are computing the evolution of a probability distribution on states of the system. At the start, this probability distribution is concentrated at a single point. As the algorithm works, the probability distribution spreads out as it becomes possible to reach far away states by a sequence of accepted moves. And eventually, the probability distribution stabilizes (we hope) as we average over an ever-increasing space of trajectories which cover the entire space Ω.

4. THE BOLTZMANN DISTRIBUTION

Equivalently, instead of studying the probability that a single run of the algorithm $x_0, \ldots, x_k$ is in a given state w after a given number of steps k, we can consider the distribution of a large collection of size N of runs started at the same x_0. If the probability of finding a single walker at state w at this point is $p(w)$, we expect to find $p(w)N$ of our large collection. In order to simplify notation, let's now assume that the states are indexed by $1, 2, \ldots, i$, and refer to the number of runs in state i at this point as n_i. In this language, $n_i/N = p(i)$. The n_i are called occupation numbers.

Observe that given a set of occupation numbers n_i, there is more than one distribution of the states occupied by our N runs which can lead to these occupation numbers. For instance, if there were N states, and each $n_i = 1$, we still wouldn't know *which* run was in *which* state, so there would be $N!$ ways this distribution of occupation numbers could occur. In general, we can compute

$$W(\{n_i\}) = \frac{N!}{\Pi_i n_i!}$$

Here we use the convention that $0! = 1$, as usual, to denote the fact that there is only one way for zero runs to occupy a given state.

Now suppose that we knew (don't ask how yet!) that the total energy of all the walkers was E_{total}. In this case, we would have two constraints on the system:

$$\sum n_i = N, \quad \sum E_i n_i = E_{\text{total}}. \tag{1}$$

We now have the following idea:

Ansatz (organizing idea). Even when restricted to the set of occupation numbers which satisfy the constraints of (1), the distribution W is so sharply peaked that the overwhelming majority of configurations have occupation numbers which maximize W subject to these constraints.

We now solve the problem of maximizing W subject to the constraints of (1), obtaining the following result:

Proposition 2. *Each value of E_{total} implicitly defines a constant T (called the temperature of the system) with the property that the distribution of occupation numbers which maximizes W has probabilities $p_i = n_i/N$ given by the* Boltzmann distribution

$$p_i = e^{-E_i/T}/Z, \text{ where } Z = \sum e^{-E_i/T}.$$

Proof. The first trick is to realize that we can maximize $\ln W$ instead of W. Now

$$\ln W = \ln(N!) - \sum \ln(n_i!)$$

Further, we have Stirling's approximation $\ln k! \simeq k \ln k - k$, which holds approximately for large k. Replacing the $\ln$ terms, we see that

$$\begin{aligned} \ln W &\simeq N \ln N - N - \sum n_i \ln n_i - n_i \\ &\simeq N \ln N - \sum n_i \ln n_i \end{aligned}$$

because $\sum n_i = N$ is one of our constraints. Now if we rewrite the first N in $N \ln N$ as $\sum n_i$, we can improve our expression for $\ln W$ to

$$\begin{aligned} \ln W &\simeq N \ln N - \sum n_i \ln n_i \\ &\simeq \sum n_i (\ln N - \ln n_i) \\ &\simeq -\sum n_i (\ln n_i - \ln N) \\ &\simeq -\sum n_i \ln(n_i/N). \end{aligned}$$

Since $n_i = N p_i$, we can last rewrite things as

$$\ln W \simeq -N \sum p_i \ln p_i.$$

Finally, since it is equivalent to maximize $(\ln W)/N$ instead of $\ln W$, we can now rewrite the entire problem in terms of the probabilities p_i:

$$\text{Maximize } f(\vec{p}) = -\sum p_i \ln p_i, \quad \text{subject to } \sum p_i = 1 \text{ and } \sum E_i p_i = E_{\text{total}}/N.$$

Now it's true that the p_i are actually all rational numbers with denominator N. But since we're approximating anyway, and assuming that N is large, we may as well think of the p_i as continuous variables. From this point of view, the optimization problem above is an exercise in Lagrange multipliers. The gradient of the objective function is given by

$$\nabla f = (-\ln p_1 - 1, -\ln p_2 - 1, \ldots, -\ln p_n - 1).$$

while the gradient of the first constraint is $(1, \ldots, 1)$ and the gradient of the second constraint is $(E_1, \ldots, E_n)$. If λ_1 and λ_2 are the Langrange multipliers, we see that at the maximum, we have for each i

$$-\ln p_i - 1 = \lambda_1 + \lambda_2 E_i, \text{ or } p_i = e^{-1-\lambda_1} e^{-\lambda_2 E_i}$$

Since $\sum p_i = 1$, we must have

$$1 = \sum e^{-1-\lambda_1} e^{-\lambda_2 E_i} = e^{-1-\lambda_1} \sum e^{-\lambda_2 E_i}.$$

This means that in particular,

$$e^{-1-\lambda_1} = \frac{1}{\sum e^{-\lambda_2 E_i}},$$

a fact that we can use to rewrite our expression for the p_i as

$$p_i = \frac{e^{-\lambda_2 E_i}}{\sum e^{-\lambda_2 E_i}}$$

The denominator in this sum is usually denoted Z and called the partition function. Now we can use the second constraint equation to write an equation involving λ_2:

$$\sum E_i e^{-\lambda_2 E_i}/Z = E_{\text{total}}/N,$$

which specifies λ_2 implicitly in terms of E_{total}, as promised. For thermodynamic reasons, the reciprocal of λ_2 is called the temperature, T of the system, and writing $\lambda_2 = 1/T$ gives the final expression

$$p_i = \frac{e^{-E_i/T}}{\sum e^{-E_i/T}}$$

which we promised above. □

5. MARKOV CHAINS

Example. Suppose that we have a very simple example where there are two states in Ω: state 1 with energy 1 and state 2 with energy 2, both states are connected by an edge, and we run the Metropolis algorithm on this system. We can encode the operation of the algorithm by a 2x2 *transition matrix* M where M_{ij} gives the probability of moving to state i from state j. A little bit of thought convinces you that the matrix M is

$$\begin{pmatrix} 1 - e^{-1/T} & 1 \\ e^{-1/T} & 0 \end{pmatrix} \quad \text{since} \begin{pmatrix} p(\text{rejection}) = 1 - M_{12} & \Delta E = +1 \\ \Delta E = -1 & p(\text{rejection}) = 1 - M_{21} \end{pmatrix}$$

If we start the algorithm with a certain probability p_1 of being in state 1 and p_2 of being in state 2, then on the next step, the occupation probabilities become

$$\begin{pmatrix} 1 - e^{-1/T} & 1 \\ e^{-1/T} & 0 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}$$

and in general if the initial vector of occupation probabilities is $\vec{p}$, on the k-th step, we have occupation probabilities $M^k\vec{p}$. Notice that each row of the matrix sums to one (since the chain must be in some state after being in state i with probability 1).

Definition 3. A *Markov chain* is the set of probability vectors given by iterating a transition matrix M (an $n \times n$ matrix, all of whose rows sum to 1) on an initial vector of probabilities $\vec{p}$. The chain is *aperiodic* if some $M_{ii} > 0$, and *ergodic* if M cannot be made block-diagonal by applying a permutation matrix[2].

We then have

Theorem 4 (Perron-Frobenius Theorem)**.** *The transition matrix M of an ergodic, aperiodic Markov chain has a unique eigenvector v_1 with eigenvalue $\lambda_1 = 1$. All other eigenvalues $\lambda_2, \ldots, \lambda_n$ have $|\lambda_i| < 1$.*

The consequence of this is that if we write the initial probability distribution $\vec{p}$ in terms of the eigenvectors of M as

$$\vec{p} = \sum a_i v_i, \quad \text{then} \quad M^k\vec{p} = \sum a_i \lambda_i^k v_i = a_1 v_1 + \sum_{i>1} a_i \lambda_i^k v_i.$$

As $k \to \infty$, the sum on the right approaches zero (since $|\lambda_i| < 1$ when $i > 1$), and we see that as long as $a_1 \neq 0$,

$$\lim_{k\to\infty} M^k\vec{p} = a_1 v_1.$$

This proves

Theorem 5. *For almost every starting configuration, a Markov chain converges geometrically to the probability distribution v_1. This is called the* stationary distribution *of the chain.*

Example. In our initial example, *Mathematica* tells us that the eigenvectors are

$$v_1 = \begin{pmatrix} \frac{e^{1/T}}{1+e^{1/T}} \\ \frac{1}{1+e^{1/T}} \end{pmatrix} \quad \text{and} \quad v_2 = \begin{pmatrix} \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

with eigenvalues 1 and $-e^{-1/T}$. As $T \to 0$, we can see that the stationary distribution is almost entirely concentrated in the minimum energy state 1, with only a tiny probability

[2] This means that we can't partition the states into two groups so that the probability of reaching one group from the other is zero.

of being in state 2. On the other hand, as $T \to \infty$, $e^{1/T} \to 1$ and the algorithm has equal probability of being in either state. This means that as we reduce $T \to 0$, the algorithm becomes increasingly likely to be in the minimum energy state, and the Metropolis algorithm succeeds at this global minimization problem!

6. CONNECTING THE METROPOLIS ALGORITHM TO THE BOLTZMANN DISTRIBUTION

We can now write down the general transition matrix for the Metropolis algorithm. Suppose that $\#n(j)$ is the number of states which are adjacent to state j. Then we have

$$M_{ij} = \begin{cases} 0, & \text{if } i \text{ and } j \text{ are not neighbors} \\ \frac{1}{\#n(j)}, & \text{if } i \text{ and } j \text{ are neighbors and } \Delta E = E(i) - E(j) \leq 0 \\ \frac{e^{-\Delta E/T}}{\#n(j)}, & \text{if } i \text{ and } j \text{ are neighbors and } \Delta E = E(i) - E(j) > 0 \\ 1 - \sum_{k \neq j} M_{kj}, & \text{if } i = j \end{cases} \tag{2}$$

This is all pretty obvious, except for the last one, which is the probability that the step was rejected and hence that we stayed in state j.

We now prove that

Proposition 6. *At any fixed temperature T, the Boltzmann distribution*

$$p_i = e^{-E_i/T}/Z, \text{ where } Z = \sum e^{-E_i/T}.$$

is the stationary distribution for the Markov chain given by the Metropolis algorithm M_{ij} of (2) *as long as the number of neighbors of each state is the same.*

Proof. We first show that if a probability distribution $\vec{p}$ has

$$M_{ij}p_j = M_{ji}p_i$$

for all i and j, then $\vec{p}$ is the stationary distribution for M. To prove this, we sum over j:

$$\sum_i M_{ij}p_j = p_j \sum_i M_{ij} = p_j.$$

and

$$\sum_i M_{ji}p_i = (M\vec{p})_j.$$

Since this is true for all j, this is really the matrix equation $M\vec{p} = \vec{p}$, which is of course exactly the statement that p is the stationary distribution.

Now we need to show that the transition matrix for the Markov chain and the probability vector given by the Boltzmann distribution have this property. Observe that if i and j are not neighbors, then $M_{ij} = M_{ji} = 0$ and the statement is trivially true. So suppose i and j are neighbors, and (wlog) that $E(i) > E(j)$. Then

$$\frac{M_{ij}}{M_{ji}} = \frac{e^{-\Delta E/T}}{\#n(j)} \frac{\#n(i)}{1} = e^{-\Delta E/T}.$$

But

$$e^{-\Delta E/T} = \frac{e^{-E(i)/T}}{e^{-E(j)/T}} = \frac{p_i}{p_j}.$$

where these are the p_i and p_j from the Boltzmann distribution. Cross-multiplying,

$$M_{ij}p_j = M_{ji}p_i$$

which is exactly what we wanted. □

7. PAUSE TO APPRECIATE OUR GAINS

We have now shown that if we give the Metropolis algorithm enough time to run at each temperature, it will eventually converge to the Boltzmann distribution, which is heavily biased toward low-energy states. Further, as the temperature continues to be reduced, we should converge to the Boltzmann distribution at temperature 0, which is guaranteed to be in the (global) minimum energy state. What could possibly go wrong?

- You cool too fast, trapping yourself in a basin which you never climb out of.
- You cool too slowly, and the universe undergoes heat death before you answer your problem.

Theorem 7 (Geman and Geman, 1984)**.** *There is a cooling schedule which is guaranteed to arrive at the global minimum energy as long as the state space is finite and connected.*

However, this cooling schedule is way too slow to be practical. In practice, one almost always uses

$$T(t) = T_0 \rho^t, \quad \text{(exponential cooling)}$$

with T_0 set relatively large and $\rho \simeq 1$ (say $\rho = 0.99$). You then cool until temperature approaches zero, or until you've reached a value of the energy function which is good enough for your purposes.

8. WORKED EXAMPLE: CODEBREAKING

Demonstration: Codebreaking.