Simulated Annealing – A Review

Aditya Chempakasseril
achempak@ucsd.edu

1. Introduction

Optimization is a field of mathematics that has applications far beyond the discipline that produced it. From finance to architecture to computer science, it finds uses in nearly all facets of the academic and professional world. With so many use cases, what good, then, is a numerical optimization technique if it is unable to optimize? Standard deterministic procedures such as gradient descent guarantee perfect results, but only when the function to be optimized is convex on the domain of interest. These results require two criteria to be met, however: first, the function needs to be convex, and second, the gradient of the function must either be known, or it must be able to be approximated. If one or both conditions isn't met, then there is no guarantee that gradient descent will succeed.

In 1983, Kirkpatrick *et al*. [1] proposed a stochastic optimization technique based on the Metropolis algorithm [2] that does not suffer from the same drawbacks as do deterministic methods like gradient descent. In fact, under certain relatively unrestrictive conditions, this method, known as simulated annealing, is guaranteed to converge. The application of simulated annealing, unlike gradient descent, is not limited to continuous domains either. In fact, its performance is most striking in combinatorial optimization problems.

This paper will summarize the concepts that are required to understand how simulated annealing was devised, followed by a description of the algorithm itself and then a few convergence results. It will end with simulation resuls.

2. Markov Chains

Before getting into the algorithm, which is actually quite simple, it's important to understand the theoretical framework underpinning its workings. This begins with a review of Markov processes (the chain of points produced by the annealing algorithm is itself such a process). A Markov process is a subclass of stochastic processes that has a foot in fields as diverse as biology and geology. In order for a process to be a Markov process, it just needs to satisfy one underlying equation, given here [3]. For all $t_1 < t_2 < \dots < t_n$,

$$P_{1|n-1}(y_n, t_n | y_1, t_1; \dots; y_{n-1}, t_{n-1}) = P_{1|1}(y_n, t_n | y_{n-1}, t_{n-1}) \qquad (1)$$

Here, $P_{1|k}$ is the probability of an event occurring given that k events have already occurred, where an event is denoted by (y_i, t_i). This equation says that, for a Markov process, the probability that some event occurs is dependent only on the event immediately preceding it.
A direct result of this property is given by Equation 2:

$$P_3(y_1, t_1; y_2, t_2; y_3, t_3) = P_1(y_1, t_1)P_{1|1}(y_2, t_2|y_1, t_1)P_{1|1}(y_3, t_3|y_2, t_2) \qquad (2)$$

The conditional probabilities on the RHS of Equation 2 are known as transition probabilities. In other words, the probability of any set of events occurring can be determined from just two facts: the probability of the first event occurring, and a list of transition probabilities that describe how one event is related to another. This powerful result is what makes Markov processes such a useful tool.

With the inclusion of two more properties, it's possible to further specify Markov processes into the class of Markov chains. These properties are that:

1. The range of Y is a discrete set of states.
2. The time variable is discrete and takes only integer values of *t*.

Examples of Markov chains include board games that are played with dice and weather predictions that depend only on the weather on the preceding day. While simulated annealing is built on the framework of Markov chains, it has been shown that even relaxing the first condition above yields an algorithm with the same convergence results [4].

When describing a Markov chain, typically the transition probabilities from Equations 1 and 2 are given by a transition, or Markov, matrix A ∈ **M**, where **M** is the set of all Markov matrices and a_{ij} represents $P_{1|1}(y_j, t_n \mid y_i, t_{n-1})$. Since each element of row *i* in A is a conditional probability relating *i* to some *j*, it follows that $\sum_{j=1}^{n} a_{ij} = 1$. A convenient byproduct of arranging transition probabilities in a matrix is the ability to easily depict them in a graph, as shown in Figure 1.

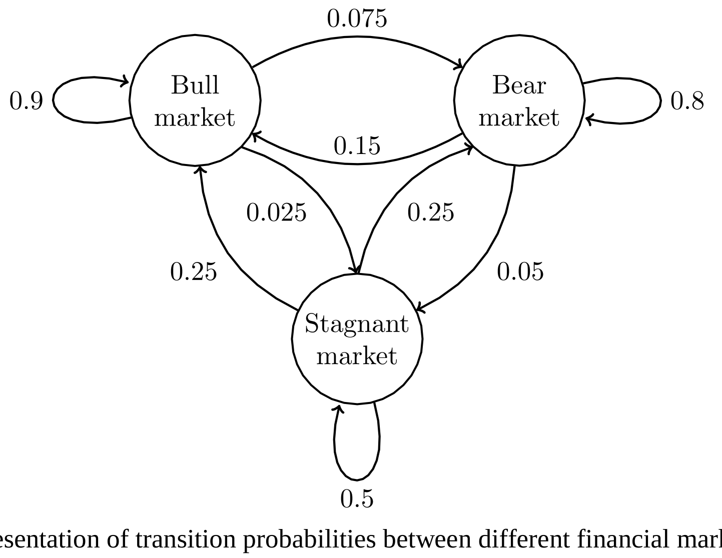

Figure 1: Graph representation of transition probabilities between different financial markets [5]. Note that the probabilities extending from any given node all sum to unity.

Of some use in this review is a brief analysis of the spectral properties of Markov matrices, as they lead to interesting results regarding the Markov chains the matrices produce. By definition of Markov matrices, it is trivial to show that the for an $n \times n$ Markov matrix, (1, **e**), where **e** is the *n*-dimensional vector of ones, will always be an eigenpair. From this, it can be shown through application of the Gersgorin circle theorem [6] that $\lambda_1 = 1$ is the largest eigenvalue for any Markov matrix.

Now, while **e** isn't particularly useful, the left eigenvector that corresponds to λ_1 speaks volumes about the matrix. The following theorem details this [6].

Theorem 1 *Let $n \geq 2$, and let $A \in \mathbf{M}_n$ be a Markov matrix with all entries positive. Then there is a unique $\mathbf{x} \in \mathbb{R}^n$ such that $A^T\mathbf{x} = \mathbf{x}$ and $\mathbf{x}^T\mathbf{e} = 1$. All entries of $\mathbf{x}$ are positive and $\lim_{p\to\infty} A^p = \mathbf{e}\mathbf{x}^T$.*

By definition, **x** is then a probability vector. Because $\lim_{p\to\infty} A^p = \mathbf{e}\mathbf{x}^T$, **x** is also known as the limiting distribution of *A*. Finally, a stationary distribution of A is defined as a probability vector **p** that satisfies $A^T\mathbf{p} = \mathbf{p}$. From Theorem 1, **x** satisfies this condition. Thus, the limiting distribution of A is also a stationary distribution of A. If A is ergodic – it is aperiodic and positive recurrent – then **x** is the only stationary distribution of A. This special distribution is of utmost importance in the next section of this paper detailing the Metropolis algorithm.

3. Metropolis-Hastings Algorithm

For those familiar with chemistry, it may be known that for chemical reactions in equilibrium, there is no net production of products or reactants. There is an identical condition for the flow of probability from one state to another state in certain Markov chains known as the detailed balance condition, the definition of which is given below:

Definition 1 *Given a Markov matrix $A \in \mathbf{M}$, a_{ij} represents $P_{1|1}(y_j, t_n|y_i, t_{n-1})$. Let $\boldsymbol{\pi}$ represent a probability distribution. The detailed balance condition states that: $\pi_i a_{ij} = \pi_j a_{ji}$*

In other words, for two states in a Markov process satisfying the detailed balance, the flow of probability between the two states is equal in both directions. If $\boldsymbol{\pi}$ and A satisfy the detailed balance condition and if the process is ergodic, then $\boldsymbol{\pi}$ is a stationary distribution of A.

With this technical machinery in place, the Metropolis-Hastings algorithm can be described. This algorithm enables sampling from a distribution, *P*(*x*), whose pdf is not explicitly known beforehand. The goal is to find some stationary distribution, $\boldsymbol{\pi}$, of a Markov chain, that is equal to *P*. First, let the detailed balance condition be rewritten as:

$$P(x')P(x^t|x') = P(x^t)P(x'|x^t) \tag{3}$$

From here, let $P(x^t|x')$ be rewritten as the product $g(x^t|x')A(x^t,x')$. Here, g is some proposal conditional distribution, and A is an acceptance ratio. It can easily be shown by substitution into Equation 3 that the following is a suitable acceptance ratio:

$$A(x^t,x') = min\left(1, \frac{P(x^t)g(x'|x^t)}{P(x')g(x^t|x')}\right) \tag{4}$$

This acceptance ratio, known as the "Metropolis choice," is the crux of the Metropolis algorithm, which follows below.

1. Pick an initial state **x'**
2. Generate a new state $\mathbf{x^t}$ based on the previous state **x'** and conditional distribution *g*
3. Calculate A($\mathbf{x^t}$**, x'**)
    a. If A = 1, then set **x'** equal to $\mathbf{x^t}$
    b. If A < 1, then generate a random number $u \in [0,1]$ uniformly. If u ≤ A, then set **x'** equal to $\mathbf{x^t}$. If u > A, then reject $\mathbf{x^t}$ and set **x'** equal to **x'**.
4. Repeat steps 2 and 3. For *n* large enough, the empirical distribution of $\mathbf{x_0, x_1,.., x_n}$, will approach *P*(*x*).

While the proof of convergence is quite complex, an intuitive reason follows. If samples are being generated with respect to the previous sample and with *g*, then they form a Markov chain, and if they are being accepted according to Equation 4, then they should form the Markov chain satisfying Equation 3. Hence, by evaluation of *P* at only some subset of the points in its domain, its entire pdf can be reconstructed.

4. Simulated Annealing Algorithm

Before describing the relationship between simulated annealing (SA) and the Metropolis algorithm, some setup is necessary. Let Ω be a very large finite set of vectors, and let $V: \Omega \to \mathbb{R}$, where *V* can have many local and global maxima. Let *V** be the maximum of *V*, and let Θ denote the set of all $x \in \Omega$ that *V* transforms to *V**. The goal of simulated annealing follows:

$$\text{Find } V^* = \max_{x \in \Omega} V(x) \tag{5}$$

Now, in light of Equation 3, what if we were able to define some distribution, *P*, that is equal to Θ? This would enable the use of the Metropolis algorithm to directly find Θ. Kirkpatrick, *et al.* did just that [1]. Let $T' > 0$, and let:

$$P_{T'}(x) = \frac{e^{T'V(x)}}{\sum_{x \in \Omega} e^{T'V(x)}} \tag{6}$$

Taking the limit as $T' \to \infty$, Equation 6 becomes:

$$\lim_{T' \to \infty} P_{T'}(x) = \frac{\delta(x, \Theta)}{|\Theta|} \tag{7}$$

where $\delta(x, \Theta) = 1$ if $x \in \Theta$ and 0 otherwise, and $|\Theta|$ is the cardinality of Θ. Hence, in the limit of high T', Equation 6, also known as Gibbs' equation, is a distribution on which SA can be applied to find the maxima of *V*. In order to use Equation 4, however, a proposal distribution *g* must be determined. In other words, given some state x_0, how can the next state x_1 be produced?

The idea here is to introduce the concept of neighboring states. x_1 can only by selected from the set of neighbors, N_x, of x_0. Of that set, the probability of any state being selected is $1/|N_x|$. Oftentimes, the set of neighbors of x_o and x_1 are similar enough that $g(x'|x^t)$ and $g(x^t|x')$ are considered equal. With this condition in place, and letting $T = 1/T'$, an acceptance ratio for the Gibbs distribution in the SA algorithm is:

$$A(x^t, x') = \min\left(1, e^{\frac{-(V(x^t) - V(x'))}{T}}\right) \tag{8}$$

In the limit of low *T*, SA using Equation 8 results in an algorithm that should theoretically jump between the maxima of *V*. In practice, this almost never happens because, like gradient ascent, the model is likely to get stuck at local maxima. A clever trick that Kirkpatrick *et al*. introduced to avoid this problem was to start running the algorithm at a high *T* and to slowly decrease it towards zero. So, initially the algorithm scans the entire domain of the function, and, as T is lowered, slowly pinpoints global maxima. This behavior is evident by the functional form the acceptance ratio.

The intuition behind SA came from metallurgy. As very hot metals are slowly cooled, they crystallize with less defects than if they were rapidly cooled. This process is known as annealing.

5. Simulated Annealing – Convergence

What good is an optimization algorithm if it never reaches convergence? For SA, convergence is equivalent to saying that : $\lim_{t \to \infty} P[x(t) \in \theta] = 1$. In 1988, Sasaki and Hajek [7] proved the following theorem, where *S* is the state space and *S** are optimal states.

Theorem 2 *We say that state i communicates with S* at height h if there exists a path in S (with each element of the path being a neighbor of the preceding element) that starts at i and ends at some element of S* and such that the largest value of V along the path is V(i)+h. Let d* be the smallest number such that every $i \in S$ communicates with S* at height d*. Then, the SA algorithm converges if an only if* $\lim_{t \to \infty} T(t) = 0$, *and* $\sum_{t=1}^{\infty} \exp[-\mathrm{d}^*/T(t)] = \infty$.

One common cooling schedule is of the form $T(t) = \frac{d}{\log t}$. Thus, as long as $d \geq d*$, Theorem 2 is satisfied and SA will converge. The problem here is picking a $d \geq d*$. This requires knowing something about the magnitude of the extrema of V a priori, which isn't always possible. A bound for the rate of convergence of SA was also produced by Sasaki and Hajek.

Theorem 3 *For any cooling schedule of the form $T(t) = \frac{d}{\log t}$, and for all t,* $\max_{x(0)} P[x(t) \notin S^*|x(0)] \geq A/t^a$ *. Here A and a are constants that depend on V and the neighborhood structure.*

Bertsimas and Tsitsiklis [8] argue that this bound is completely unhelpful as the constants in the equation are extremely large, dominating the asymptotic behavior. Furthermore, they claim that for any fixed T, the algorithm performs worse than a random walk over the state space. This begs the question: why would anyone ever use simulated annealing?

Bertsimas and Tsitsiklis go on to explain that while optimal solutions $V*$ may take very long for the algorithm to arrive at, close to optimal solutions, $\hat{V}$,have been found very quickly in practice. This is most evident in combinatorial optimization problems such as the graph partitioning problem, the graph coloring problem, and the traveling salesman problem [8].

6. A Simulation

While SA isn't particularly fit for continuous optimization, it was tested here on the Rastrigin function, given by:

$$f(\boldsymbol{x}) = An + \sum_{i=1}^{n} [x_i^2 - Acos(2\pi x_i)] \tag{9}$$

Figure 2 shows a visualization of this function in three dimensions.

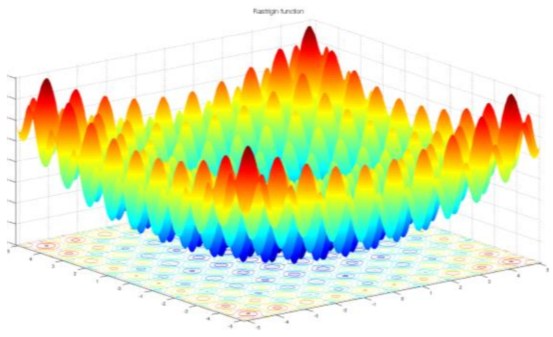

Figure 2: 3-dimensional visualization of the Rastrigin function [9]. Note the difficulty finding a global minimum.

Equation 9 is a test function designed explicitly for optimization algorithms. Note the number of local minima in Figure 2. The global minimum of the function occurs at **x** = **0**.

The problem was run in 25-dimensional space with an initial temperature of 10 and a domain of [-100, 100] for each dimension. The initial function value was 80565.13. Notice that even in three dimensions on a domain of [-10, 10], this is a very difficult optimization problem.

A few intelligent modifications were made to the algorithm to induce faster convergence. First, the basic simulated annealing is run multiple times with a tolerance of 0.01 instead of just once with a lower tolerance. In each run, the algorithm is seeded with the best solution produced by the previous run. The reason for doing this is that the at very low temperatures, the algorithm stops searching for better minima. By restarting the algorithm with a high temperature, SA can continue to search for better minima if there are any. This “high temperature” is always slightly less than the starting temperature of the previous run to account for the fact that the current best solution is most likely closer to the global minimum than the previous best solution.

A second change follows the recommendation of the MATLAB implementation of SA [10]. In particular, the temperature schedule is $T(t) = T_0(0.97)^t$ instead of $T(t) = \frac{d}{\log t}$. After testing both, the former schedule was found to lead to significantly faster convergence. Furthermore, a modified acceptance ratio, suggested by MATLAB, is used:

$$A(x^t, x') = \min\left(1, \frac{1}{1 + e^{\frac{V(x^t) - V(x')}{T}}}\right) \qquad (10)$$

Again, this acceptance ratio led to faster convergence than that in Equation 8. The results of the simulation are given in Figures 3. As shown, after only 2000 iterations (0.296 seconds), the algorithm converged to a solution *very* close to zero (600). In comparison, gradient descent would have little hope of reaching even near the global minimum as it would most likely immediately get stuck at a local minimum.

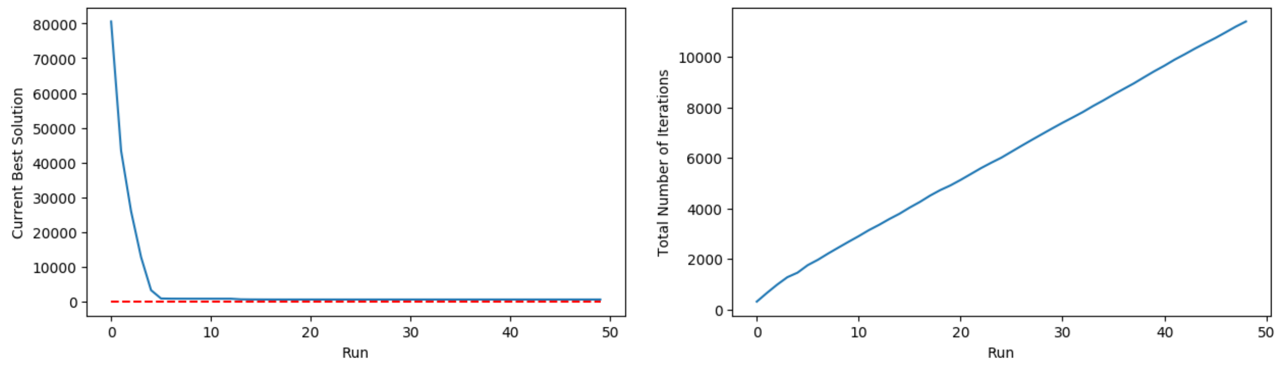

Figure 3: The chart on the left shows the progress of the SA algorithm with respect to the number of times it was restarted (the number of “runs”). The chart on the right displays the total number of iterations throughout the course of all runs.

Link to GitHub repository: https://github.com/achempak/Simulated-Annealing

7. Works Cited

1. Kirkpatrick, S., Gelatt C. D. & Vecchi, M. P. (1983), *Optimization by Simulated Annealing*, Science, DOI: 10.1126/science.220.4598.671

2. Metropolis, N., Rosenbluth, A., Rosenbluth, M. & Teller, E. (1953), J. Chem. Phys.

3. Van Kampen, N. G. (2015), *Stochastic Processes in Physics and Chemistry*.

4. Gelfand, S. B. & Mitter, S. K. *Simulated Annealing.*

5. Markov Chain. URL: https://en.wikipedia.org/wiki/Markov_chain

6. Garcia, S. R. & Horn, R. A. (2017), *A Second Course in Linear Algebra*.

7. Sasaki, G. & Hajek, B. (1988), *The Time Complexity of Maximum Matching by Simulated Annealing*, J. Assoc. Comput. Mach.

8. Bertsimas, D. & Tsitsiklis, J. (1993), *Simulated Annealing*, Statistical Science

9. Test Functions for Optimization. URL: https://en.wikipedia.org/wiki/Test_functions_for_optimization

10. Simulated Annealing Options. URL: https://www.mathworks.com/help/gads/simulated-annealing-options.html