Simulated Annealing

Suppose we have a space S and a function $f : S \to \mathbb{R}$ and that we want to find $s_{min} \in S$ such that

$$f(s_{min}) = \min_{t \in S} f(t).$$

To each element $s \in S$ there is an associated cost, $f(s)$, which we are trying to minimise.

One method is to perform a local search. Consider a sequence of elements $s_0, s_1, ..., s_n \in S$ chosen such that

$$f(s_0) > f(s_1) > \cdots > f(s_n)$$

and among these we hope that $s_n = s_{min}$. This sequence is chosen in the following manner.

1. Start by making an educated guess and choose $s_0 \in S$.
2. Given s_k, try to make small changes to s_k and transform it into an element with smaller cost, s_{k+1}. Repeat this step as long as there are ”good” changes to make, and the stop.
3. Output the current element as the one minimising f over S.

This strategy has the disadvantage of being very sensitive to the initial solution s_0, a good choice may lead to a global minimum cost solution, while a bad choice will lead to a local minima.

In order to escape from o local minima a lot of proposals have been made, some deterministic some randomised. Simulated annealing is one of those randomised proposals.

The basic idea

The general idea with simulated annealing is to perform a local search, but sometimes make changes leading to a increase in cost. As the algorithm make progress the tendency to move to worse solutions decrease, and in the end we move to better solutions only.

An analogy in a random walker trying to find the lowest point in an area by walking around randomly with the following strategy. In the beginning the walker just moves around randomly and do not care if the trail leads up or down. As times passes the walker becomes more and more lazy (tired perhaps) and is not that willing to go up any more. The enthusiasm for climbing becomes weaker and weaker, and in the end he or she only moves downhill. Hopefully the walker has reached the lowest point and stops.

An example: Graph coloring

Let $G = (V, E)$ be a graph and let $C = \{c_1, ...c_q\}$ be a set of colors. A coloring $\xi \in S^V$ is an assignment of colors to each vertex in the graph. A valid coloring is one where no vertices having the same colors are sharing an edge. Assume that there are enough colors so there exists at least one valid coloring. Let us use simulated annealing to solve the graph coloring problem.

Given a coloring we count the number of edges in E having different colors in its endpoints and let that represent the cost of a coloring. We start with an arbitrary coloring, and change it by choosing a vertex at random and giving it a random color.

State (search) space: We let S^V be the set of coloring we use for the algorithm.

Cost (energy) function: Let $f : S^V \to \mathbb{R}$ defined by the following be the cost function.

$$f(\xi) = \sum_{\langle x,y\rangle \in E} I_{\{\xi(x)=\xi(y)\}}(x,y)|$$

Initial coloring: Let X_0 be a coloring chosen from S^V uniformly at random.

Transition mechanism: Given X_n do the following.

(i) Update X_n to X_n' by choosing a vertex $v \in V$ uniformly at random, and assign a randomly chosen color to it.

(ii) Let $X_{n+1} = X_n'$ with probability $p(T)$ where

$$p(T) = \begin{cases} 1, & \text{if } f(X_n') < f(X_n) \\ e^{-(f(X_n')-f(X_n))/T}, & \text{otherwise} \end{cases}$$

Important : When creating the update mechanism it is important that the resulting Markov chain is both irreducible and aperiodic otherwise the Markov chain theory (which the algorithm is based on) cannot ensure that we will have convergence towards a stationary distribution.

The idea now is to simulate a Markov chain by starting in X_0 and update according to the update mechanism presented above. We start with a large value of T, namely T_1, and run the simulation for N_1 steps, then we change and let $T = T_2$ and run the chain for another N_2 steps, and so on. For this we need a cooling schedule.

A cooling schedule: A cooling schedule determines how fast the tendency towards worse coloring decay. It consists of two parts. First we need a sequence of decreasing temperatures $(T_1, T_2, ...)$ such that

$$\lim_{n\to\infty} T_n = 0.$$

Second we need an increasing sequence of running times $(N_1, N_2, ...)$. This mean that we run $(X_0, X_1, ...)$ by using temperature T_1 for $(X_0, X_1, ..., X_{N_1-1})$ and temperature T_2 for $(X_{N_1}, X_{N_1+1}, ..., X_{N_1+N_2-1})$ and so on.

Simulated annealing in practice

When running the simulated annealing algorithm in any real world application we need a more informative cooling schedule. The ”real world” cooling schedule consists of four parts.

(i) An *initial temperature*, T_0. We need to ensure that the simulated annealing walk on the state space behaves like a random walk. The temperature has to be large enough to make even the worst solutions possible. How large it needs to be has to do with how large the differences in cost are between neighbours in the state space.

(ii) A *decrement function*, that is, something telling us how to generate the next temperature. One example is $T_{n+1} = \alpha T_n$, where a suitable α usually is in the interval $[0.8, 0.99]$.

(iii) A *final temperature*. At some time point we need to stop and output a result. The finial temperature could be fixed in advance, or calculated based on the chain $(X_0, X_1, ..., X_n)$ up to the present time n.

(iv) A sequence of finite running times, one for each temperature.