Simulated Annealing

Siddharth Namachivayam

May 7

1 Introduction

The term 'simulated annealing' was first coined by Kirkpatrick, Gelatt, and Vecchi in 1983. They proposed a combinatorial optimization algorithm inspired from experimental methods used in physics to analyze the 'ground-state' (i.e. minimum energy level) of a system by first heating it up and then letting it slowly cool down to low temperatures. Simulated annealing is a Monte Carlo method because at each temperature 'the system' is represented by a target probability distribution we want to generate draws from. In this project, we will outline the simulated annealing algorithm, provide some intuition as to why it works, and implement it in R to try and solve the travelling salesman problem.

2 The Boltzmann Distribution

Say we are trying to minimize the function E over some finite domain X. Let $M = \min_{x \in X} E(x)$. Then at temperature T, the simulated annealing algorithm wants to draw from the distribution:

$$p_T(x) = \frac{e^{\frac{M-E(x)}{T}}}{Z_T}$$

where:

$$Z_T = \sum_{x \in X} e^{\frac{M-E(x)}{T}}$$

In statistical mechanics, p_T is known as the Boltzmann distribution. Why do we want to draw from it? Note as we cool:

$$\lim_{T \to 0} e^{\frac{M-E(x)}{T}} = \begin{cases} 1 & x \in E^{-1}(M) \\ 0 & x \notin E^{-1}(M) \end{cases}$$

So:

$$\lim_{T \to 0} Z_T = \lim_{T \to 0} \sum_{x \in X} e^{\frac{M-E(x)}{T}} = \sum_{x \in X} \mathbb{I}(x \in E^{-1}(M)) = |E^{-1}(M)|$$

Thus the distribution we are drawing from approaches:

$$\lim_{T \to 0} p_T(x) = \frac{\mathbb{I}(x \in E^{-1}(M))}{|E^{-1}(M)|}$$

, i.e. the uniform distribution over the set of minimizers for E!

It is sensible to ask how we are supposed to draw from p_T if we don't know the value of M. However, if we use Metropolis-Hastings to draw from the distribution, note the relevant terms in the acceptance ratio:

$$\frac{p_T(y)}{p_T(x)} = \frac{e^{\frac{M-E(y)}{T}}}{e^{\frac{M-E(x)}{T}}} = e^{\frac{E(x)-E(y)}{T}}$$

cause the contributions of $e^{\frac{M}{T}}$ to cancel out.

Here is a cool visualization of some draws from the Boltzmann distribution at high, medium, and low temperatures for an arbitrary energy landscape E:

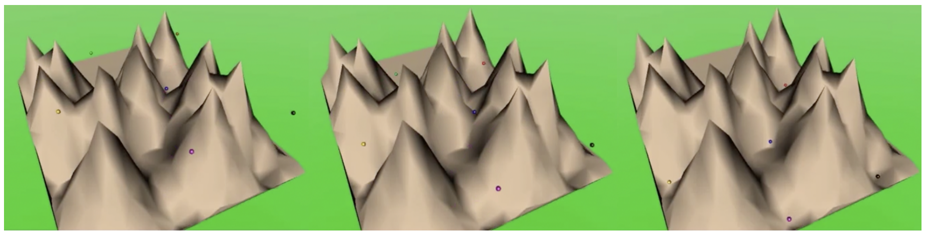


Figure 1: Source- https://www.youtube.com/watch?v=KQYfaitQn7g

3 The Homogeneous Algorithm

Simulated annealing can be implemented as either a homogeneous algorithm or an *in*homogeneous algorithm. A homogeneous algorithm can be "described by a sequence of homogeneous Markov chains" (Laarhoven and Aarts 1988) whereas an inhomogeneous algorithm cannot. A Markov chain $X_0, X_1, X_2, ...$ is homogeneous iff $\forall n \in \mathbb{N}$:

$$\mathbb{P}(X_{n+1} = j \mid X_n = i) = \mathbb{P}(X_n = j \mid X_{n-1} = i)$$

In the homogeneous algorithm for simulated annealing, the temperature is slowly decreased according to a cooling schedule $T(k)$ for the kth Markov chain. The kth Markov chain is a Metropolis-Hastings chain which draws from the Boltzmann distribution $p_{T(k)}$ defined in section 2. When the temperature is sufficiently close to 0, we stop running chains and return our estimated minimizer.

Note the algorithm is homogeneous because each Metropolis-Hastings chain is by construction homogeneous!

Algorithm 1 Homogeneous Simulated Annealing

```
1: procedure HOMO_ANNEAL($x$, $T$, steps)
2:     $k \leftarrow 1,\ s \leftarrow x$
3:     while $T(k) > 0.001$ do                              ▷ Temp is not sufficiently small
4:         $s \leftarrow$ boltzmann_mh_chain($s$, $T(k)$, steps)     ▷ Draw from $k$th chain
5:         $k \leftarrow k + 1$
6:     return $s$                                          ▷ The estimated minimizer is s
```

4 The Inhomogeneous Algorithm

Let's now look at the inhomogeneous algorithm. In this case, the temperature is slowly decreased according to a cooling schedule $T(k)$ for the kth *step* in a single inhomogeneous Markov chain. The Markov chain is similar to a Metropolis-Hastings chain, however in the kth step if $Y_x = y$ is drawn given $X_{k-1} = x$, then we draw a standard uniform U and set $X_k = Y_x$ if:

$$U \leq \frac{p_{T(k)}(y)q(y,x)}{p_{T(k)}(x)q(x,y)} = e^{\frac{E(x)-E(y)}{T(k)}} \cdot \frac{q(y,x)}{q(x,y)}$$

or $X_k = X_{k-1}$ otherwise.

Algorithm 2 Inhomogeneous Algorithm Step

```
1: procedure STEP_INHOMO_ANNEAL($x$, $y$, $k$)
2:     $u \leftarrow$ runif(1)
3:     $r \leftarrow e^{\frac{E(x)-E(y)}{T(k)}} \cdot (q(y,x)/q(x,y))$
4:     if $u \leq r$ then                                  ▷ Check if draw less than MH ratio
5:         return $y$
6:     else
7:         return $x$
```

When the temperature is sufficiently close to 0, we stop running the chain and return our estimated minimizer.

We have to be particularly careful with our cooling schedule in the inhomogeneous algorithm since if we cool too quickly in *real* annealing "the substance is allowed to get out of equilibrium...[and may form] only metastable, locally optimal structures" (Kirkpatrick et al. 1983). The homogeneous algorithm keeps 'the substance' in equilibrium by only returning draws from the *stationary* distribution of each Metropolis-Hastings chain.

The inhomogeneous algorithm effectively accomplishes this by using an *extremely* slow cooling schedule. The most popular schedules (Hajek 1988) are of the form:

$$T(k) = \frac{T_0 \cdot \log(2)}{\log(1+k)}$$

, where T_0 is the starting temperature. The idea is analogous to that of quasistatically compressing a box of gas with a piston so as to not produce excess entropy by knocking the gas out of equilibrium. In practice, this is done by decreasing the volume reasonably slow, so that "the gas has time to continually equilibrate to the changing conditions" (Schroeder 2000).

Here's the pseudo-code for inhomogeneous simulated annealing:

Algorithm 3 Inhomogeneous Simulated Annealing

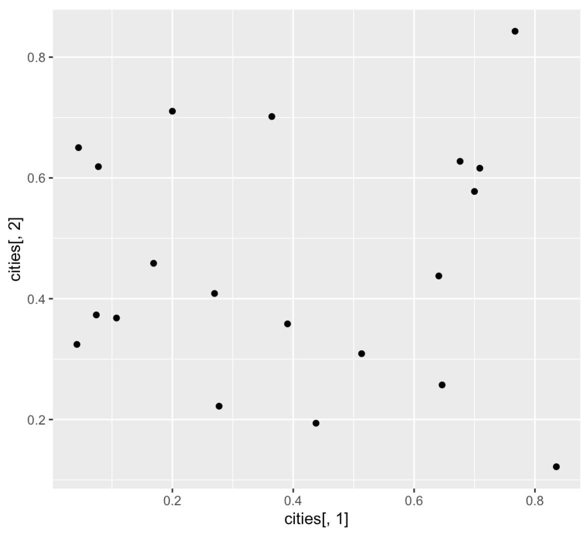
```
1: procedure INHOMO_ANNEAL($x$, $T$)
2:     $k \leftarrow 1,\ s \leftarrow x$
3:     while $T(k) > 0.001$ do                ▷ Temp is not sufficiently small
4:         $y \leftarrow m(s)$                ▷ Generate proposal state conditional on $s$
5:         $s \leftarrow$ STEP_INHOMO_ANNEAL($s$, $y$, $k$)
6:         $k \leftarrow k + 1$
7:     return $s$                             ▷ The estimated minimizer is s
```

5 Travelling Salesman Problem in R

In the travelling salesman problem, we are given a set of cities and want to create a round trip that visits all of them so that the total distance we travel is minimized. This is an NP-hard problem so naive approaches like the greedy algorithm will converge too quickly. However, since the space of all possible paths we could take is finite, this is a combinatorial optimization problem we can apply simulated annealing to! In particular, we'll implement the inhomogeneous algorithm with a cooling schedule of the form presented above in section 4.

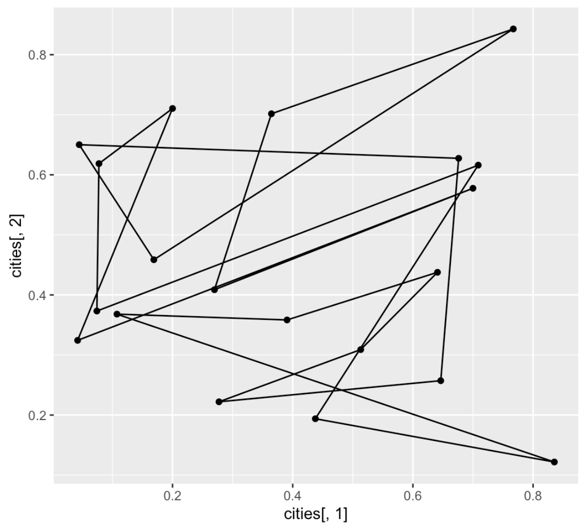
Let's first generate the problem we want to solve:

~~~
```{r}
library(tidyverse)
library(knitr)
library(raster)
n <- 20
cities <- matrix(runif(2*n), ncol=2)
g <- ggplot()+geom_point(aes(cities[,1], cities[,2]))+coord_fixed()
g
```
~~~



Our starting point in the chain will be:

~~~
```{r}
x <- sample(1:n)
path <- rbind(cities[x,], cities[x,][1,])
g <- g+geom_path(aes(path[,1],path[,2]))
g
```
~~~



Similar to Metropolis-Hastings, we need a way to make moves in the path-space so we can propose new states for our chain. Since each path is uniquely determined by an arrangement of the cities, we can simply transpose two cities at random to propose a new path:

~~~
```{r}
m <- function(path) {
  x <- path[-nrow(path),]
  y <- x
  flips <- sample(nrow(x), 2)
  y[flips[1],] <- x[flips[2],]
  y[flips[2],] <- x[flips[1],]
  y <- rbind(y,y[1,])
  return(y)
}
```
~~~

The function we are trying to minimize is:

~~~
```{r}
path_length <- function(path) {
  d <- 0
  for (i in 1:n) {
    d <- d + pointDistance(path[i,], path[i+1,], FALSE)
  }
  return(d)
}

path_length(path)
```
## [1] 8.362617
~~~

Since our moves form a symmetric random walk, our 'Metropolis-Hastings' step is actually more of a $\mathrm{MR}^2\mathrm{T}^2$ step:

~~~
```{r}
step_inhomo_anneal <- function(x, y, k){
  u <- runif(1)
  temp <- log(2)/log(1+k)
  r <- exp((path_length(x)-path_length(y))/temp)
  if(u<r){
    return(y)
  }
  else{
    return(x)
  }
}
```
~~~

Now we can implement simulated annealing:

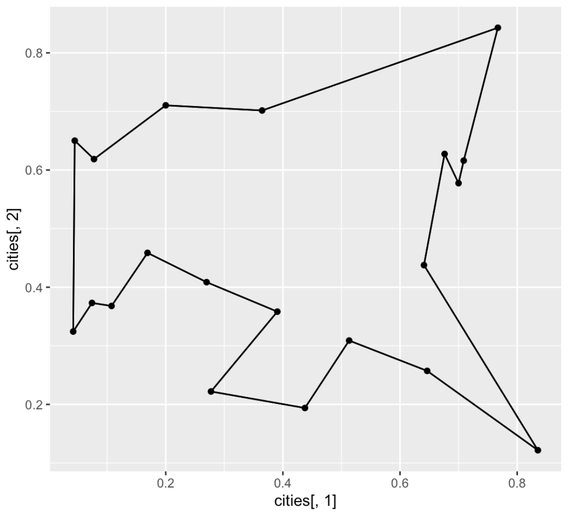
~~~
```{r}
inhomo_anneal <- function(x, threshold){
  k <- 1
  s <- x
  temp <- log(2)/log(1+k)
  while(temp > threshold){
    y <- m(s)
    s <- step_inhomo_anneal(s,y,k)
    k <- k+1
    temp <- log(2)/log(1+k)
  }
  return(s)
}

path <- inhomo_anneal(path,0.07)
print(path_length(path))
```
~~~

```
## [1] 3.306822
```

That's pretty good considering we started with a path length ≈ 8. Let's take a look at the route:

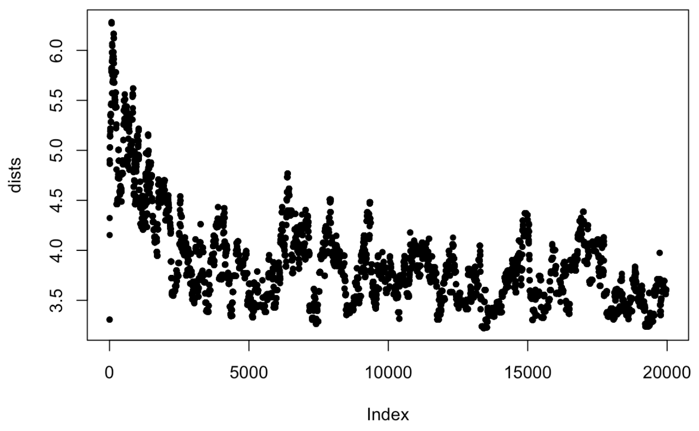
~~~
```{r}
g <- g+geom_path(aes(path[,1],path[,2]))
g
```
~~~



Here's a plot of the path lengths as we raise and then cool the temperature again over time:

~~~
```{r}
inhomo_anneal <- function(x, threshold){
  k <- 1
  s <- x
  temp <- log(2)/log(1+k)
  dists <- c()
  while(temp > threshold){
    dists <- c(dists, path_length(s))
    y <- m(s)
    s <- step_inhomo_anneal(s,y,k)
    k <- k+1
    temp <- log(2)/log(1+k)
  }
  return(dists)
}

dists <- inhomo_anneal(path,0.07)
plot(dists,pch=20)
```
~~~



Looks like the algorithm is doing what we wanted!

6 Conclusion

Simulated annealing is a Monte Carlo algorithm inspired from statistical mechanics used to solve combinatorial optimization problems. Intuitively, the algorithm creates an energy landscape using the function we are attempting to minimize and then slowly cools the system down to its 'ground-state.'

The algorithm comes in two main flavors. The first is a sequence of homogeneous Markov chains which generate draws from the Bolztmann distribution at progressively lower temperatures. The second is a single inhomogeneous Markov chain which approximates the first algorithm by decreasing the temperatures *extremely* slowly in between subsequent steps.

As seen above in section 5, simulated annealing can be used to generate relatively good solutions to NP-hard questions such as the travelling salesman problem. Our rudimentary implementation used a starting temperature of $T_0 = 1$ with a cooling schedule:

$$T(k) = \frac{T_0 \cdot \log(2)}{\log(1+k)}$$

and a threshold $T(k) > 0.07$. With further improvements in running time and alternate cooling schedules, we can increase T_0 and lower our threshold to yield even higher quality solutions.

7 Works Cited

1. Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. In Science, Vol. 220, No. 4598. pp. 671-680.

2. X Code. (2007). Simulated Annealing. www.youtube.com/watch?v=KQYfaitQn7g.

3. Van Laarhoven, P. J. M., and L. Aarts. (1992). Simulated Annealing: Theory and Applications. Kluwer Academic Publishers.

4. Hajek, B. (1988). Cooling Schedules for Optimal Annealing. In Mathematics of Operations Research, Vol. 13, No. 2. pp. 311–329.

5. Schroeder, D. V. (2000). An Introduction To Thermal Physics. Addison Wesley.