Monte Carlo Markov Chains for the Traveling Salesman Problem

1 Notations

- $\otimes$: multiplication matricielle standard
- $\times$: multiplication case à case
- ${}^{[i]}_{j}$: neurone j de la couche i du réseau de neurones
- f : fonction d'activation, il s'agit ici de la fonction sigmoïd : f(x) $= {}^{1}/{}_{1+e^{-x}}$
- L : erreur calculée à la sortie du réseau de neurone. La fonction erreur appropriée dans le cas d'une sigmoïd est la log-vraissemblance

2 Introduction

The Traveling Salesman Problem is an NP-hard optimization problem, asking the question : "Given a list of cities with their coordinates and a starting point, what is the shortest path that visits each city and returns to the starting point ?".

It is theoretically possible to explore all possible path and save the best one. But in practice, the complexity of such an algorithm will explode in computing time but also in memory allocation if the number of given cities grows.

Since the trivial algorithm exploring all potential routes is not viable, many other algorithms have been developed to find at least one of the optimal solutions.

The Simulated Annealing algorithm is one of these algorithms, and will be explained in details in this report.

3 Exploring all possibilities

4 The Simulated Annealing Algorithm

Let the initial route where σ_0 is the starting city :

$$\sigma^{[0]} = (\sigma_0, \sigma_1, ..., \sigma_m) \tag{1}$$

We compute the length of a route with the following formula :

$$H(\sigma^{[i]}) = \sum_{k=0}^{m-1} \delta(\sigma_k^{[i]}, \sigma_{k+1}^{[i]}) + \delta(\sigma_m^{[i]}, \sigma_0) \tag{2}$$

Where δ is a function which compute the length between both cities given as parameters.

The idea of the Simulated Annealing Algorithm applied to the Traveling Salesman Algorithm is to minimize the system's energy. Such an energy is here described as the length of the current selected route. Then, in order to minimize this energy, the target is to find the shortest route.

As explained above, doing so by exploring all routes and selecting the shortest one become impossible. The solution here is to propose a random neighbour of our current configuration, and see if choosing it makes the system's energy decreasing.

We say that two different configurations are neighbours if it is possible to reach one of them from the other by swapping two of its cities.

For example, with 4 cities given, these both configurations below can be called neighbours :

$$\underbrace{\sigma^{[i]} = (\sigma_0, \sigma_3, \sigma_1, \sigma_2) \quad ; \quad \sigma^{[k]} = (\sigma_0, \sigma_1, \sigma_3, \sigma_2)}_{Neighbours} \tag{3}$$

Then, from a current configuration, we can choose a random neighbour if it has smaller length than the current configuration, but if not, we can still choose it if it has a greater probability of acceptance than a random value u following an uniform law between 0 and 1.

This probability of acceptance is described as :

$$P(\sigma^{[i]}, \sigma^{[k]}, T) = e^{-1/T \times (H(\sigma^{[k]}) - H(\sigma^{[i]}))} \tag{4}$$

Where $\sigma^{[i]}$ is the current configuration, $\sigma^{[k]}$ its randomly selected neighbour, and T the temperature of the system.

The temperature T has its whole importance here. Indeed, higher the temperature is, closer to 1 will be the probability of acceptance, In opposite lower the temperature is, closer to 0 will be the probability of acceptance. Furthermore, bigger is the gap between $H(\sigma^{[i]})$ and $H(\sigma^{[k]})$, lower will be the probability of acceptance.

Such a mechanism avoids being stuck in a local optimum, permitting to move to a neighbour even if it makes the global energy increase. However, too big temperature causes the probability of acceptance stuck at 1. The proposed configurations will always be accepted in this case. In opposite, with a very small temperature the process will never choose a bigger energy configuration, which increase the risk of being stuck into a local optimum.

Decreasing continuously the temperature is part of the Simulated Annealing Algorithm. The system will explore a lot of different configurations at the beginning, accepting configurations with a big length, but more the process will be iterated, more the probability of acceptance will decrease. At this point the system will slowly stabilize itself around small length configurations.

The game is to choose a good initial temperature, and choose a pretty larger number of iteration in order to let the system cooling down slowly. In these conditions, a global optimum with a very low length should be found, so the probability of acceptance should be pretty small since it will be potentially hard to find a new configuration which is as good as the current one.

Here is an example of function which can be used for decaying the temperature :

$$T_{i+1} = T_i \times 0.99 \tag{5}$$

Such a decaying temperature can remind us the decaying learning rate for a neural network back-propagation.

4.1 Pseudo-Code

Algorithm 1 Simulated Annealing

```
1:  Function : Route
2:  Inputs :
3:      $\sigma^{[0]}$ : Initial Route
4:      $T_0$ : Initial temperature
5:      $n$ : Number of iteration
6:      $coef$ : Shrinking coefficient
7:  Output :
8:      $\sigma$ : Optimum Route found
9:  Local variables :
10:     $\sigma^{'}$ : Route
11:     $T$ : Current temperature
12:     $i$ : Iteration number
13: Beginning :
14:     $T \leftarrow \mathrm{T}_0$
15:     $\sigma \leftarrow \sigma^{[0]}$
16:     For $i$ in 1 to $n$, do :
17:         $\sigma^{'} \leftarrow$ randomNeighbour($\sigma$)
18:         If $H(\sigma) > H(\sigma^{'})$ , then :
19:             $\sigma \leftarrow \sigma^{'}$
20:         Else :
21:             If $P(\sigma, \sigma^{'}, T) > \mathcal{U}(0,1)$, then :
22:                 $\sigma \leftarrow \sigma^{'}$
23:             End If
24:         End If
25:         $T \leftarrow \mathrm{T} \times coef$
26:     End For
27:     Return $\sigma$
28: End
```

4.2 Strengths

As explained above, computing all the different routes and keeping the best one is absolutely not feasible. The Simulated Annealing Algorithm is a pretty good option in order to find one of the best configuration of the system with much less computation time and memory allocation.

4.3 Weaknesses

The major problem of this algorithm is choosing a good initial temperature and the perfect decaying rate. These both parameters permit to avoid a potential local optimum and to prevent from jumping out from the global optimum configuration. The value of these parameters are hard to choose, forcing us to repeat the algorithm many times in order to keep the best inputs.

A golden rule is to choose these parameters in order to make the temperature cooling down to zero.

Furthermore, like a lot of other algorithms coming from graph theory, is it impossible to theoretically prove that the best configuration found is the global optimum.

4.4 Two different final configurations

Both final configurations printed below have been generated with these initial parameters :

- Initial temperature : $T_0 = 5000$
- Shrinking coefficient : $coef = 0.9999$
- Number of iterations : $n = 350000$

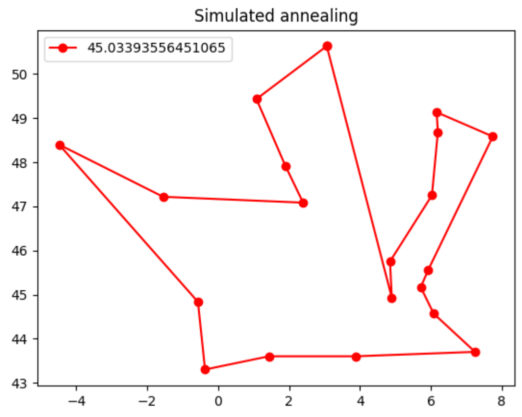

FIGURE 1 – *Final Temperature :* $\approx$ *45.03*

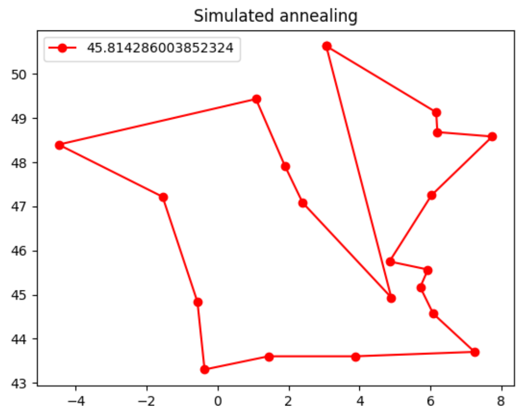

FIGURE 2 – *Final Temperature :* $\approx$ *45.81*

Here is a pretty satisfying *GIF* animation showing the evolution of the system :
mega.nz/#!60lDjTwR