Simone Quadrelli

Genetic algorithm Monte Carlo

Academic year 2019 - 2020

Contents

List of Tables

List of Figures

Abstract

The aim of this project is to solve the travelling salesman problem (TSP) which consists in finding the shortest cycle among all the cities. The problem is a *NP-hard* problem and therefore there exist no feasible algorithm to solve if for any possible set of cities in input. The appromximately optimal solution was computed by the simulated annealing, while the possible cycles are produced by a genetic algorithm.

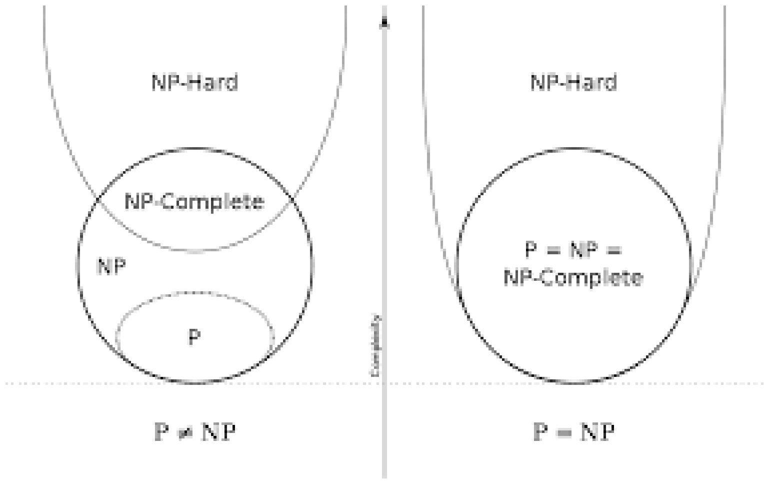
1 Introduction

The travelling salesman problem consists in finding the shortest route a travelling salesman has to take to visit all the city he must reach only once and to return to the starting point. The route described is called in graph theory *hamiltonian cycle* (i.e a cycle that pass through each vertex of a graph just once), indeed the oldest and more technical formulation of the problem was proposed by Hamilton.
An algorithm that provides a solution to TSP can be exploited in a wide variety of applications: indeed it can be used in logistics to optimize the path of vehicles and therefore to reduce costs. It is possible to model the problem usign an undirected, weighted and fully connected graph [1] $G = (N, E)$, where N is the set of the cities to visit and $E = N \times N$ is the weight of the path. In this particular instance of the probelm the weights correspond to the euclidean between each couple of cities. THe mailtonian cycle can be defined as a sequence $(n_0, .., n_{k-1})$ of length k such that $n_i \neq n_j \; \forall i, j \in \{1, ..., |N|\}$ and such that $(i, j) \in E \; \forall i, j \in \{1, ..., |N|\}$ The most naive solution is to compute all the possible paths but they are factorial in the number of cities n in input. The number of possible cycles is $(n-1)!$ and grows faster that 2^n (i.e $2^n = o(n!)$). Therefore even if it was possible to find them in constant time the algorithm will have an expnential time complexity and would not be solved in finite time for large n. More techically, TSP is an NP-hard problem. A NP-hard problem is a problem at least as hard as the hardest NP-complete problem. NP-complete problem is a problem that can be solved in polynomial time by a non-deterministic algorithm. For such a class of problem there exists no polynomial algorithm that can solve the problems and may not exist, still noone was able to prove that such algorithms does not exists. Bearing in mind the complexity problem to face, the most reasonable solution is to solve the problem approximately. The technique I to generate the possible hamiltonian cycles is a genetic algorithm, while the choice of the fittest hamiltonian cycle is done by the simulated annealing algorithm

2 Simulated annealing

Simulated annealing (SA) is a technique that mimic the behaviour of heated materials in metallurgy that can be slowly cooled down, this process decreases the impurity and increases the size of the crystals. It is a technique that pproximates the global optimal solution when the solutions is to be searched in an enormous space. SA is a metaheuristic meaning that it is not a probelm specific heuristic and that can be used to reduce the search space of an algorithm, the genetic algorithm in our case.
SA very usefull since it does not require any prior information, indeed it just supposes that the

[1] For an hamiltinian cycle to exist the graph must be at least connected



states (hamiltonian cycles in our case) are distributed accordingly to the Bolzmann distribution:

$$\exp^{-\beta(cost(newstate)-cost(oldstate))} \tag{1}$$

The temperature T stated before is $T = \frac{1}{\beta}$, therefore we start with low values of betas (high temperature) and as the simulation proceeds we increase the values of β, thus mimicing the decrease in temperature. The only other information needed is the cost function. For this specific application the cost function is the euclidean distance between the cities. The (x, y) are expressed in latitude and longitude (CONTROLLA L'ORDINE). Therefore given the coordinates of two cities (x_1, y_1) and (x_3, y_2)

$$cost((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{2}$$

```
function SIMULATED ANNEALING
    s ← generate starting state
    for b in betas do
        for n in 1,.., number of simulations do
            ns ← sample new states with metropolis algorithm
        end for
    end for
    return ns
end function
```

3 Metropolis algorithm

Metropolis is a Markov Chain Monte Carlo (MCMC) algorithm used to generate values X_n from a finite set of values S accordingly to a target probability distribution π.
An homogeneous and finite Markov chain with state space S, transition matrix P and initial distribution μ is a sequence of random variables $\{X_n\}_{n \in N}$ such that

1. $\forall i \in S \, Pr(X_0 = i) = \mu(i)$
2. $\forall n > 0 \, \forall i_n, ..., i_0 \, Pr(X_{n+1} = j | X_n = i_n, ..., X_0 = i_0) = Pr(X_{n+1} = j | X_n = i)$
3. $\forall n > 0 \, Pr(X_{n+1} = j | X_n = i) = p(i, j)$

Metropolis algorithm is designed in such a way that it is very lilely to move from a state with lower probability from a stete with higher probability.
The following algorithms describes the metropolis algorithm when the target distribution is $\pi(\text{state}) = \exp^{\beta \text{cost(state)}}$ and S is the set of all possible hamiltonian cycles having fixed their length. In the following we suppose to have generated two hamiltonian cycles (i.e states) old_state, new_state and we want to understand if new_state can come from π. If new_state is likely to be a sample of π we use it as new state otherwise we keep old_state as the current state.

```
function METROPOLIS(old_state, new_state)
    if cost(new_state) < cost(new_state) then
        return new_state
    end if
    threshold ← unif[0,1]
    p ← $min\{1, \exp^{-\beta(cost(new\_state) - cost(old\_state))}\}$
    if p > threshold then
        return new_state
    else
        returm old_state
    end if
end function
```

3.1 Properties of metropolis algorithm

We can model the generation of new hamiltonian cycles usign a undirected and connected graph $G = (S, E)$ whose set of states S contains all the possible hamiltonian cycles and E contains the transition probability from an hamiltonian cycle to another one. Given the graph, we can define the transition matrix $P = p(i, j)]_{i,j \in S}$ as

$$p(i,j) = \begin{cases} 0 & \text{if } \{i,j\} \notin E \\ \frac{1}{d_i} \min\{1, \frac{\pi(j)}{\pi(i)}\} & \text{if } \{i,j\} \in E \; i \neq j \\ \frac{1}{d_i} \sum_{l \in adj(i)} \min\{1, \frac{\pi(l)}{\pi(i)}\} & \text{if } i = j \end{cases} \tag{3}$$

Since G is a connected graph, the P is irriducible meaning that $\forall i, j \in S \, p^n(i, j) > 0$, where $p^n(i, j)$ means that exist a path from i to j of length n.
It can also be proved that P is aperiodic [1]. By aperidoicity we mean each node of the graph has a loop. More formally, th graph G is aperiodic if $\forall i \in S$, $d(i) = 1$ where

$$d(i) = mcd\{n \in N | \, \text{exist a cycle from i to i of length} \, n\} \tag{4}$$

Metropolis algorithm defines a probability distributin π that is a reversible distribution, meaning that

$$\pi(i)p(i,j) = \pi(j)p(j,i) \;\; \forall i,j \in S \tag{5}$$

This is a key feature of the algorithm since a reversible distribution π is also a stationary distribution $\pi' P = \pi'$. This condition holds by the following theprem

Theorem 1. *If a probability distribution π is a reversible distribution for a markov chain $\{X_n\}$, then it is also a stationary distribution.*

Proof. $(\pi' P)_j = sum_{i \in S} \pi(i)p(i,j) = \sum_{i \in S} \pi(j)p(j,i) = \pi(j)$ □

If this distribution was unique than we are certain to correctly generate the data accordingly to the target distribution. The stationary distibution is unique if the transition matrix is primitive as stated by the follwing theorem.

Theorem 2. *Let $\{X_n\}$ be a markov chain with a primitive (i.e primitive and aperiodic) transition matrix on a finite set, then there exist only one stationary distribution and it as also the limit distribution* $\lim_{x \to \infty} p^n(i,j) = \pi(j) \, \forall i \in S$

Since P is primitive is enough to compute the limit distribution to find the stationay distribution. The stationary distribution can be approximated in a logarithmic number of iterations starting from any distribution.Therefor to sample hamiltonian cycle from the stationary distribution is enough to let the simulation go on for a suitable number of iterations.

4 Genetic Algotirthm

Genetic algorithms are a widespread class of algotirhms that mimic nature. Indeed there are three fundamental operations they perform, each operation is inspired by the theory of evolution by Darwin. Among a specie evolution selects the fittest individuals and therefore the algorithm will chose the fittest (i.e shortest) hamiltonian cycles. The breeding of the fittest individuals produces new individuals that inherits traits from the parents, therefore in the crossbreeding phase we mix some subsection of the fittest hamiltonian cycles. As random mutations can occur in nature but they are not so likely to accur, the algorithms swaps two nodes (cities) with a fixed and low probability. My own implementation of the algorithms performs the selection operation as follows:

```
1: function SELECTION(population,n_paths)
2:     n_parents ← unif[2,n_paths/2]
3:     order the population by fitness
4:     fittest_subpopulation ← extract n_parents in order from the sorted population
5:     return fittest_subpopulation
6: end function
```

where **population** is an array of hamiltonian cycles

```
function CROSSBREEDING(fittest_subpopulation)
    for i in 1, ..., n_paths do
        choose parent1 uniformely among the fittest_subpopulation
        Choose parent2 uniformely among the fittest_subpopulation
        if parent1 different from parent2 then
            choose position1 randomly among unif([0,length of cycles-1)
            choose position2 randomly among unif([0,length of cycles-1)
            divide parent1 in three sections:
            [parent10:position1]
            parent1[position1:position2]
            parent1[posotion2:]
            divide parent2 in three sections:
            parent2[0:position1]
            parent2[position1:position2]
            parent2[posotion2:]
            breed child1 justapposing in the order:
            parent1[0:position1]
            all the cities not in parent1[0:position1] and not in parent2[posotion2:]
            parent2[posotion2:]
            breed child2 justapposing in the order:
            parent2[0:position1]
            all the cities not in parent2[0:position1] and not in parent1[posotion2:]
            parent1[posotion2:]
        else
            child1 ← parent1
            child2 ← parent1
        end if
    end for
    new_population is a vector of the children
end function
```

2) Crossbreeding −¿ obtain new hamiltonian cycles exchanging the gines of the parents 3) Mutation −¿ exchange genes randomly

```
function MUTATE(new_population, p)
    test ← unif[0,1]
    if test < p then
        choose position1 randomly among unif([0,length of cycles-1)
        choose position2 randomly among unif([0,length of cycles-1)
        swap the cities in position1 and position2
    end if
    order the population by fitness
    fittest_subpopulation ← extract n_parents in order from the sorted population
    return fittest_subpopulation
end function
```

```
function EVOLVE(population)
    fittest_population ← SELECT(population)
    new_population ← CROSSBREED(fittest_population)
    population ← MUTATE(new_population)
end function
```

5 dataset

6 results

References

[1] Oliver Catoni *Simulated annealing algorithms and Markov chains with rare transitions*, Sminaire de probabilits (Strasbourg), tome 33 (1999), p. 69-119.

[2] G. E. Hinton and R.R. Salakhutdinov, *Reducing the Dimensionality of Data with Neural Networks*. Science, 29 July 2006, Vol 313, 504-507.