# Laboratorio di Fisica Computazionale
# Exam Report

Giulia Tranquillini

giulia.tranquillini@gmail.com

Course of AA 2020-2021 - MSC in Theoretical Physics

## The Traveling Salesperson Problem solved with Simulating Annealing Methods

## 1 Introduction

The well-known *Traveling Salesperson Problem* (TSP) is part of the combinatorial optimisation cathegory of problems in which the purpose is to find the global minimum of a function dependent from one or more parameters.
The discussion of this pages has started with the random collocation of $N$ points on a squared grid of side $L = \sqrt{N}$. These points corresponded to different cities organised on a map: the purpose of this report was to implement a simulation to identify the shortest route which comprehended all the cities, visiting each of them one and only one time. The quantity involved in the minimisation of the overall crossed distance $D$, and so the parameter to be modified for finding the solution, was the $N$-dimensional array `travel`, which contained the sequence of cities identified with labels in the interval $[1, N]$.
The program `salesman_problem.f90` was written in `Fortran90` and `Gnuplot` from scratch: his main constituents were two arrays $X$ and $Y$ containing the coordinates of the cities, and a symmetric $N$ x $N$ matrix in which each point $(i, j)$ corresponded to the distance between the $i-th$ and $j-th$ city. The traveled distance $D$ is considered directly proportional to a "cost" function which we will call $E$ with similarity with the energy of a physical system, the parameter of which we usually want to find the lower bound.
In the following images several initial map configurations for $N = 4, 8, 20, 50$ are reported: all of them are created by the -

```
subroutine place_cities(initial_distance, X, Y)
```

The output of the subroutine is an `initial_distance` calculated from the random disposition and denominated as $D_0$ (each time a different seed is used in order to increase uncorrelation between different runs).
The cities labels depend on the random sampling of couples of coordinates and will obviously remain unchanged in the whole simulation.

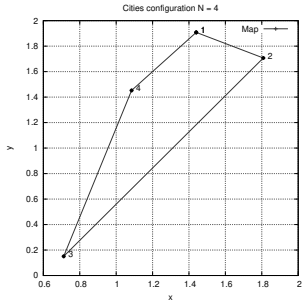


Figure 1: $N = 4, D_0 = 4.255$

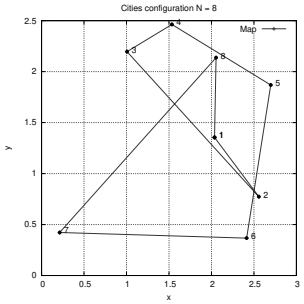


Figure 2: $N = 8, D_0 = 11.814$

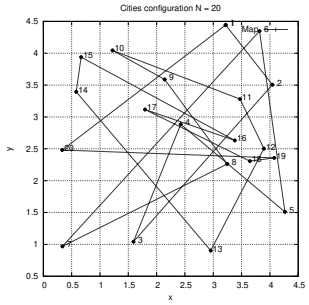


Figure 3: $N = 20, D_0 = 46.177$

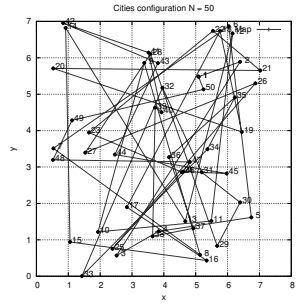


Figure 4: $N = 50, D_0 = 182.671$

The solution of the TSP will be reached in three different ways: in the first place a *Brute Force Algorithm* will be performed, and his limitations will be discussed. Then the quest will be solved with a *Monte Carlo Metropolis - Simulating Annealing Procedure*, and in the end a *Demon Simulating Annealing Algorithm* will be implemented. After that, for each case, a particular emphasis will be given to the convergence time of the algorithm.
Due to computational power issues (see the conclusion paragraph) the calculation of the error for th $D_{min}$ is left for a possible future more detailed analysis, in my case performing more runs ($\simeq$ 30min each for high $N$) in order to have the possibility to manipulate a statistic set of values has been impracticable.

# 2 Methods Implementation

## The Brute Force Algorithm

The benefit of using this really expensive method is that the probability of falling in a local minimum is quite low; this, combined with a visual representation of the salesman route, allowed me to find an exact solution and use it as benchmark for the following part on the study.
The heart of this algorithm is the subroutine implemented to switch pieces of the trail and calculate the lenght of the new route: two indexes are randomly chosen in the array `travel` -

```
index1 = 1 + FLOOR(N*rnd(1))
index2 = 1 + FLOOR(N*rnd(2))
```

so that the visits to two cities are interchanged.
In the program, the choice is to perform a number of permutations equal to $N!$ in order to have enough iterations to statistically find the correct absolute minimum. The complexity will unfortunately have an order of $O(N!)$: finding the exact solution for $N > 12$ has been really expensive; I decided to stop for $N = 14$ in this first section. In the following table the results for some $N$ are reported; as we can see, the convergence time is polinomially increasing:

| $N$ | $D_0$ | $D_{min}$ | $Time[s]$ |
|---|---|---|---|
| 4 | 4.255 | 4.255 | 0.00109 |
| 5 | 6.8119 | 4.622 | 0.00142 |
| 6 | 7.7162 | 5.050 | 0.00931 |
| 7 | 10.335 | 6.825 | 0.0170 |
| 8 | 11.814 | 8.097 | 0.0465 |
| 10 | 18.878 | 9.402 | 0.294 |
| 12 | 24.681 | 11.574 | 43.793 |
| 14 | 34.681 | 12.666 | 274.114 |

Table 1: Results

A better understanding of the problem is possible looking at the solution for $N = 14$: observing the array `travel`, it evolves from his initial configuration to a new one generated by the algorithm going from -

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

- to -

| 1 | 13 | 9 | 3 | 14 | 10 | 7 | 4 | 8 | 12 | 5 | 2 | 11 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

At the same time the plotted route will go from a chaotic/random one, to an optimised arrangement.

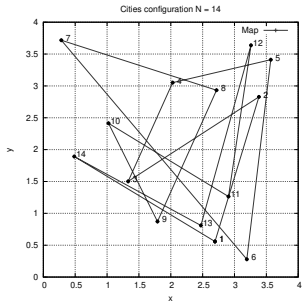


Figure 5: Initial configuration

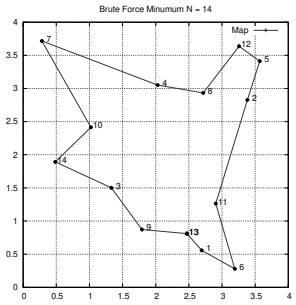


Figure 6: Final configuration

## The Simulating Annealing Algorithm

To perform the procedure of the *Simulating Annealing*, our system is compared to a thermal one, in which the temperature $T$ has an important role. As we said before, the energy of the system (the cost of the trip) is proportional to the total distance travelled. Knowing that, and wanting to keep the *Acceptance Ratio* around 0.5, we choose the initial temperature as:

$$T_0 = \texttt{initial\_distance}$$

In similarity with the cooling procedure of a crystal, $T_0$ has been constantly lowered until a lower boundary ($T < 0.1$) with steps of 0.1. The program has been tested also with steps of 0.01 but the computational cost was too high compared to the improvement in the accuracy.
At this point, for each value of $T$, a *Monte Carlo Metropolis Algorithm* has been implemented to calculate the optimal configuration. As usual, for each step, the system was perturbed and the distance was recalculated.

After that, if $\Delta E < 0$ the new configuration would be accepted, because brought the system in a less energetic state. If $\Delta E > 0$ the Boltzmann factor has been considered, and the new configuration had ben accepted with a probability $P(E,T) = exp(-\Delta E/T)$. In the program:

$$\Delta E = D_{new} - D_{old}$$

An accepted configuration was considered in the search of the absolute minimum of the problem, if rejected the program immedeately generated a new one. As in the brute force algorithm the `exchange` subroutine has been used, and a step of the MCM method was defined as the variation of a single pair of the elements of the `travel` array. The parameter `nmcs`, accounted for the number of Monte-Carlo steps, has given some problems: knowing that the configurations are $N!$ in total, the number of iterations needed to be very high; after several trials it was setted to $\texttt{nmcs} = N \cdot 10^6$

We have to remember that in this case, the probabilty of finding the correct value of $D$ is $\simeq (N-1)/N!$. For $N = 12$ we obtain that the probability of finding a route of lenght $D_{min} = 11.574$ is $2 \cdot 10^{-8}$! To stress the importance of `nmcs`, we now will analyse the case for $N = 12$ to see that a small number of repetitions will not bring the system in his point of minimum.
In the following histogram the values of the minimum reached after $\texttt{nmcs} = 10^6$ are reported in function of the temperature. From the *Brute Force Method* we found that the expected result (in yellow) is:

$$D_0 = 11.574$$

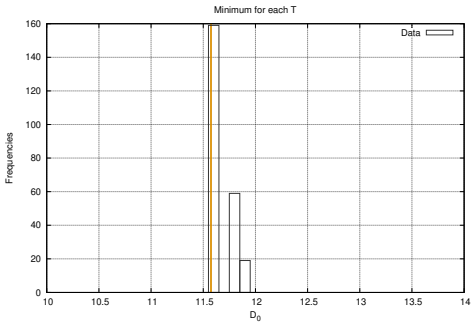


Figure 7: $N = 12$, $nmcs = N \cdot 10^6$

As we can see, the majority of the simulations ends up in the correct bin, although a longer one would have produced a still better result. To understand how the system develops, let us consider only one random value of $T$: $T_0 - n \cdot (0.1)$ and the sequence of iterative perturbations. In the following gif are reported in sequence all the minimum configurations from the initial one to the absolute minimum one.

Link to gif: $->$ Successful search.

Changing now the value of `nmcs` to a much lower one ($10^2$), we will see that no simulations ends up in the correct minimum. Starting from a $D_0 \simeq 25$ the results after *nmcs* iterations for each $T$ are more and more correct and less dispersed increasing the parameter.

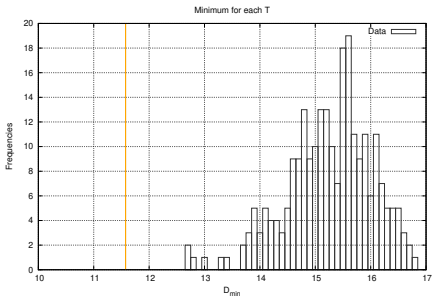


Figure 8: $nmcs = 10^2$

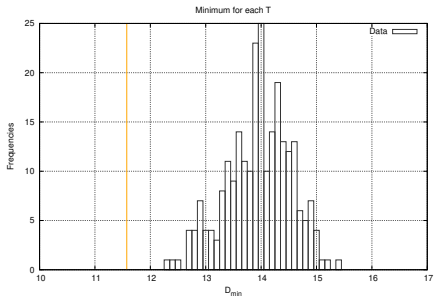


Figure 9: $nmcs = 10^3$

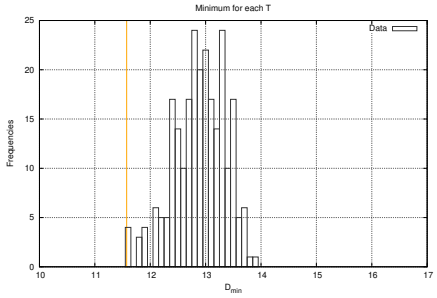


Figure 10: $nmcs = 10^4$

Looking at the animation we can see that the final route is not the shortest one, althought more "ordered" that the initial setting.

Link to gif: $->$ Unsuccessful search.

In the implementation I decided not to consider an equilibration phase, given that the initialising subroutines and the exchange one both has an intrinsic randomness yet scorrelating the various runs.

## The Simulating Annealing with Demon

As third method, the Demon Simulating Annealing has been implementeted: a Monte Carlo procedure was reproposed with a new degree of freedom: the *Demon*. The `demon` is a fictitious positive quantity with an initial value of $\sqrt{N}/4$ (see. [3]) which receives and yields energy to the system during the simulation.
The `demon` takes the place of the temperature to simulate the annealing and slowly decreases with 0.1 steps until a lower boundary of 0.5. The procedure consist of a Monte Carlo algorithm with some new mechanisms:

- Start with a random initial configuration,
- Randomly exchange the order of two `travel` columns,
- Compute the energy difference $\Delta E$,
- If $\Delta E < 0$the configuration is accepted and the system gives the energy to the demon ($E_d = E_d - \Delta E$),
- If $\Delta E > 0$ and $E_d > \Delta E$, the configuration in accepted and the system takes the energy from the demon ($E_d = E_d - \Delta E$),
- If the demon has not sufficent energy, the configuration is rejected.

In the program at each iteration ($\texttt{nmcs} = N \cdot 10^5$), a check has been done to control that the sum of the demon plus the energy of the system was constant. We now report some graphics considering only $N = 10$. With this number of repetitions the final configuration is the correct one as we can see in Figure [12].

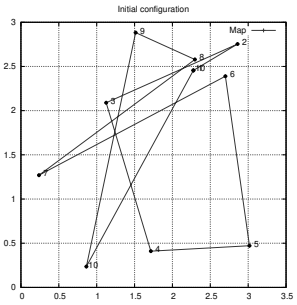


Figure 11: Initial configuration

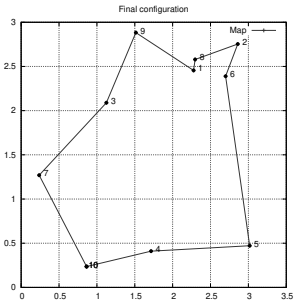


Figure 12: Final configuration

Also the histogram with the minima for each temperature shows the success of the simulation: all the values foll in the bin of the actual minimum = 9.402:

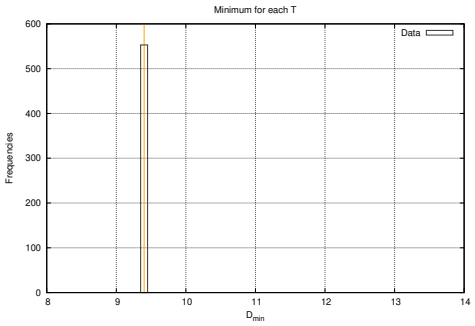


Figure 13

In conclusion both the MC variations have been tested with some higher $N$ to further test their effectiveness. All the simlations has been done with $nmcs = 10^5$ which took quite a lot of time, but the results are satisfactory.

| $N$ | $D_0$ | $D_{min}^{SA}$ | $D_{min}^{DSA}$ |
|---|---|---|---|
| 12 | 24.681 | 11.574 | 11.614 |
| 24 | 60.946 | 28.214 | 30.408 |
| 48 | 180.013 | 116.530 | 114.203 |

Table 2: SA and DSA results

# 3 Analysis and Interpretation

After the description of the three implemented methods, is interesting to insert a parenthesis on the question of time. The $TSP$ is known to be a a *NP-hard* problem, so the running time of the procedure is at least polynomial (super-polynomial) and with increasing $N$, as we see before, the cost increases excessively. We said that the Brute Force Algorithm should be abandoned for relatively low $N$, but also the following two methods have their limits.
In order to compare the performances of the methods, some runs have been mede for $N = [6, 8, 10, 12, 14]$, and the CPU-time has been calculated after `nmcs` $= N!$ iteration (this parameter has been consider the same for all the algorithms in order to test them in the same conditions).
The results shown in the following table ($BF$ = Brute Force, $SA$ = Simulating Annealing, $D$ = Demon Simulating Annealing) are expressed in $[s]$ :

As expected, considering only the procedure inside the loop, the Demon Simulating Annealing is the most expensive method. This is probably due to the fact that while the Brute Force method only has to generate new configurations and calculate the absolute minimum, the MC Simulating Anneling also has to

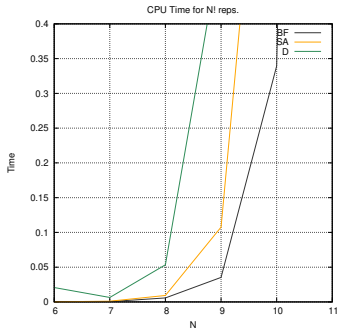


Figure 14: Time for nmcs = $N!$

add an *Accept-Reject* procedure, while the Demon Simulating Annealing have to add and subtract a real quantity from the `demon` too. So the highest number of actions slow the loop. This analysis has been made only on a little sample of $N$, so to confirm this trend it should be broaden to higher $N$.

A further analysis to compare the two different SA procedures has been done minimising the distance for $N = 10$ and `nmcs` $= N \cdot 10^5$ and plotting the energy in function of the number of iterations (expected result in [3]). The behaviour is

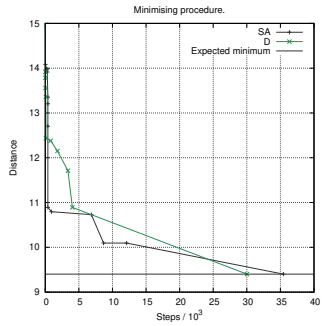


Figure 15: Distance in function of time

in accordance to the one outlined in the article, as we can see, the MC Algorithm with the Demon converges to the correct one 5000 steps before the Monte Carlo Metropolis method.

# 4 Conclusions

In conclusion is possible to say that the *Salesman Problem* has been effectively studied and solved here, in his simplest formulation, given the available time and computational power; obviously there is a huge room for improvement.
Starting from a more careful choice for the initialising parameters, we saw that also the possibility to make use of a number of iterations as near as possible to $N!$ (without an eccessive cost) would for sure increase the resulting precision.
Being this an open problem with several multidisciplinary applications (from matter physics simulations and genetic algorithms to satellite networks (see. [4])), a wide literature is accessible with improving techniques and approaches to maximise the efficiency of the implementation.
Finally, I only wanted to rapidly outline that the importance of the *Traveling Salesman Problem* is not only linked to Classical Computing. In the recent years it has also been studied with Quantum Computing techniques ( see. [5]) were possibly part of the time-cost issues are solved (with the Grover Algorithm we obtain a $O(\sqrt{N!})$ problem), and the algorithm has been adapted for gradually increasing $N$, therefore for bigger systems as molecules or ensamble of molecules.

# References


[1] H. Gould, J. Tobochnik, "An Introduction to Computer Simulation Methods Applications in Physical System", 2016

[2] I. Wood, T. Downs, "Demon Algorithms and their Application to Optimization Problems"

[3] H. Guo, M. Zuckermann, "A Fast Algorithm for Simulating Annealing"

[4] R. Matai, S. Singh, "TSP: An Overview of Application, Formulations and Solution Approaches", Department of Mechanical Engineering, Malviya National Institute of Technology Jaipur, India, 2016

[5] D. J. Moylett, N. Linden, "Quantum speedup of the Travelling Salesman Problem for bounded-degree graphs", Quantum Engineering Technology Labs, University of Bristol, UK, 2016