Simulated Annealing algorithm for solving the Traveling Salesman Problem (TSP)

Giovanni Varutti

December 24, 2023

Contents

1 Introduction and code description

The aim of this work is to write a code that applies the *Simulated Annealing* algorithm, following a Monte Carlo approach, in order to solve the well-known *Traveling Salesman Problem (TSP)*, possibly finding the optimal route for the traveler. The code and all the related material can be found inside this *GitHub* repository. The whole software is composed by some files: a *FORTRAN90* module with some supporting subroutines, the main program, also written in *FORTRAN90*, which performs the actual arguments and collects the parsed arguments, a python notebook for managing data and plots, and some bash scripts to run the program automatically for a certain configuration of parameters. Moreover, *openMP* was used for parallelizing some loops or other embarassingly parallel tasks, and the python module *Gurobi*, in order to calculate the exact optimal routes, for small configurations of nodes. The program generates an initial random arrangement of N nodes (or cities) in a square of linear dimension $L = \sqrt{N}$, using an array of dimension $(N, 2)$ to store the x and y coordinate of each node, and an array of dimension (N, N) to store all the distances. The state of the system, which is the route represented by a sequence of nodes, is stored in another 1D array, and its length is associated with the energy of an imaginary thermal system. In order to generate new random rearrangements of a route (that are the neighbours of the system's state), three different *local search* algorithms have been implemented:

- **Swap algorithm**: choose two nodes at random and to interchange the order of visit
- **Insert algorithm**: choose one node at random and insert it in a new random position of the route (rescaling all the other nodes)
- **Inverse algorithm**: choose two nodes at random, and invert the order of visit of all the nodes between them

In the following section an analysis of this algorithm will be presented, in particular for the time scalability, the results reached, and the parameters analysis.

2 Time scalability and parameters analysis

Firstly, we can show the behaviour of the algorithm with two small configurations, with 24 and 48 nodes respectively. In particular, the first configuration with $N = 24$ nodes was performed with an initial temperature of $T = 100.0$, an annealing factor of 0.95, and 1000 equilibration steps. In the second configuration, the parameters were $T = 200.0$, annealing factor of 0.99, and 10000 equilibration steps. We can plot the final routes reached in both cases by the algorithm:

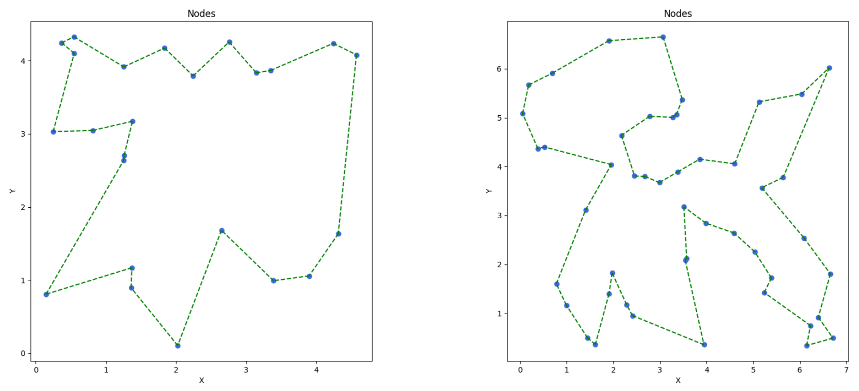

Figure 1: final state of the sistem with parameters ($N = 24$, $T = 100.0$, 0.95, 1000) and ($N = 48$, $T = 200.0$, 0.99, 10000)

In both cases the algorithm reached the optimal solution, as confirmed running the same configurations with *Gurobi*. In particular, the algorithm have reached the optimal route for the traveller in 100% of cases, for configuration with $N = 8, 12, 24$ nodes. For configurations with $N = 48$ nodes, or more, the optimal solution is reached most of the times, but not always (even testing several annealing schedules). A possible solution for this problem is to introduce a *restart* procedure, in which the annealing algorithm is repeted starting from the final configuration of a previuos annealing cycle. We can show also the trend of the system's energy (displaying all the energies of the accepted configurations), in function of the computational step.

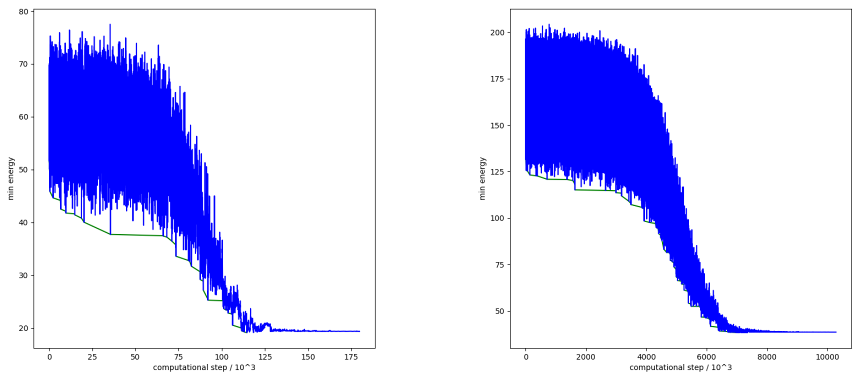

Figure 2: trend of the system's energy for the same configurations above. The green line shows only the **minimal** energies found, whereas the blue line shows all the energies of the **accepted** configurations

As expected, the blue line is quite noisy, since the algorithm accepts also changes with higher energies, with a certain probability. Moreover, using the data about the accepted states energies, we can make histograms of all the energy values reached by a system during the annealing algorithm. This histogram can then be used to determine the probability of finding a route (so a state of the system, characterized by a certain value of the energy) of a given length, for a given annealing schedule, i.e. for a given combination of the annealing algorithm's parameters. Below are reported the two histograms, related to the previous configurations:

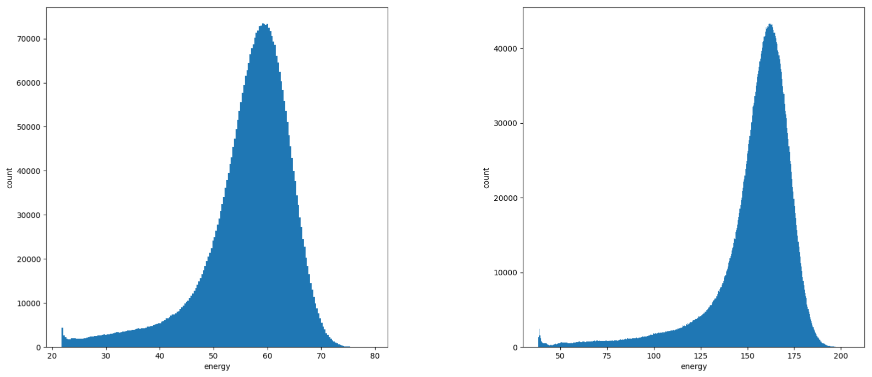

Figure 3: histograms of the energies reached by the system. The plot on the left, for the configuration with $N = 24$ nodes, has been made with 100 bins, the one on the right, for the configuration with $N = 48$ nodes, with 500 bins

Now we can do a more sophisticated analysis of the annealing schedule's parameters (which are the temperature, the annealing factor and the equilibration steps), in function of the CPU time, required to perform the whole algorithm, and the relative percentage improvement, with respect to the initial random configuration. The following plots show the processor time required to run the whole algorithm (Intel core vPRO i7 with 12 physiscal cores boosted to 5.1 GHz). For each configuration tested, the time was taken five times, and the average is then plotted. The number of nodes for all these tests is $N = 8, 12, 24, 48, 96$. The initial value of the temperature has been choosen with the same order of magnitude of the initial energy of the system's state, which depends on the number of nodes.

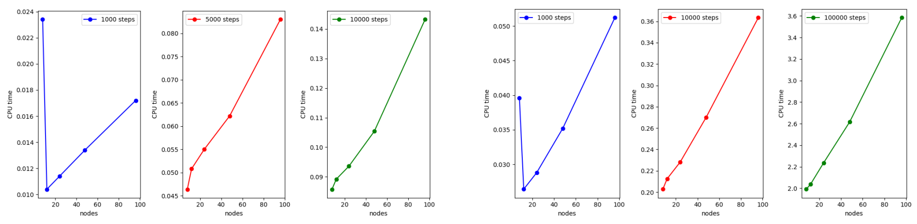

Figure 4: CPU time required by the algorithm. The plot on the left is referred to an annealing factor of 0.91, the one on the right to an annealing factor of 0.95

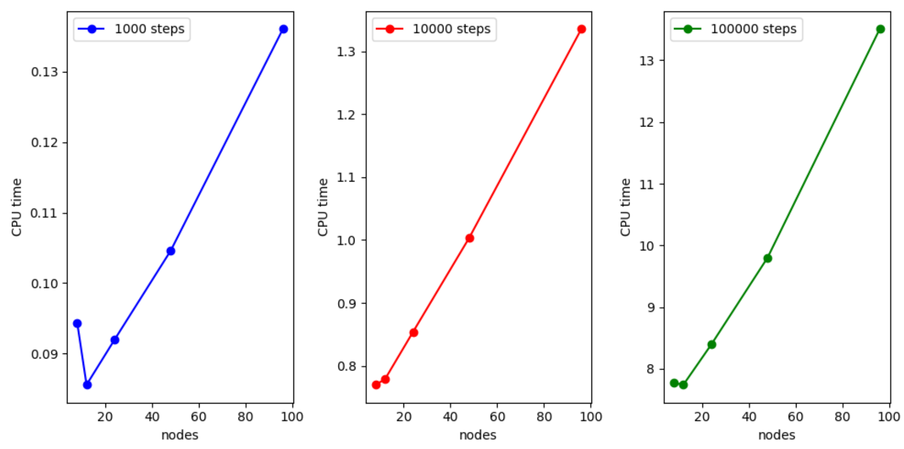

Figure 5: CPU time required by the algorithm. The plot on the left is referred to an annealing factor of 0.99

As we can see, the time scales linearly with the number of nodes for all the tested configurations. The outlier point is, for all the configurations, the one relative to $N = 8$ nodes. Moreover, we can see that, for all the annealing factor tested (0.91, 0.95, 0.99), if the number of equilibration steps increases by one order of magnitude then also the time does so. This is obviously what we expect. The next analysis takes into account the relative percentage improvement between the initial random state and the final one reached by the algorithm. In this case, the relative improvement has been measured in function of the equilibration steps, for different values of the annealing factor. Then, the procedure has been repeted for configurations with different number of nodes, precisely $N = 24, 48, 96, 192$. Also in this case, for each configuration, the improvement has been computed five times, and then the average was taken. The initial value of the temperature has been choosen approximately with the same magnitude of the initial energy, so $T = 100$ for $N = 24$, $T = 200$ for $N = 48$, $T = 500$ for $N = 96$ and $T = 1000$ for $N = 192$.

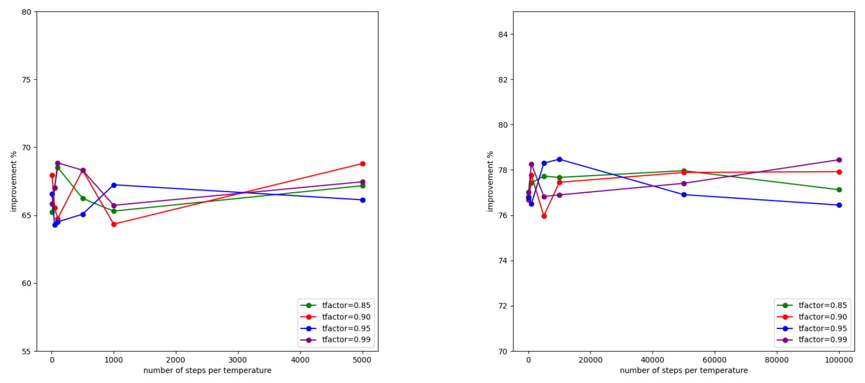

Figure 6: relative percentage improvement for configurations with $N = 24$ (left) and $N = 48$ (right)

As we can see, the relative improvement has approximately a constant behaviour, for all the values tested (both for the annealing factor and the equilibration steps). This is probability because, for small configurations with few number of nodes, the algorithm reaches the optimal solution quite easy, and the little noise in the lines is due to the value of the energy of the initial random route.

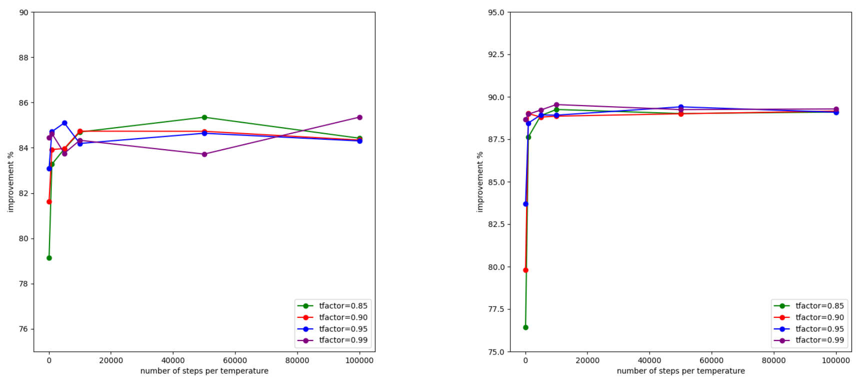

Figure 7: relative percentage improvement for configurations with $N = 24$ (left) and $N = 48$ (right)

In this case we can see, instead, a fast growth of the relative improvement when passing from 100 to 1000 equilibration steps. Then the trend saturates in a constant behaviour as before. This is why, for bigger configurations (with more nodes), 100 equilibration steps is not enough for the system to reach the quasi-equilibrium.

In the **data** folder and in the **plots** folder, all the produced material can be found.