TP4: Ant-System

BY MARTINO FERRARI

1 The Travelling Salesman Problem

Once again we will study the Travelling Salesman Problem and for this reason I will reuse the introduction I made for the last TP.

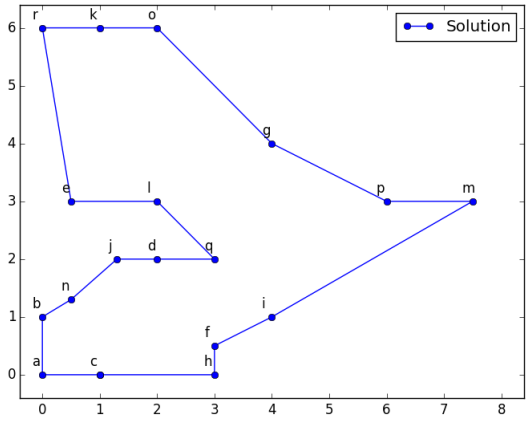


Figure 1. example of a TSP problem

The travelling salesman problem (**TSP**) is a classic and well-known problem, that consists of finding the shortest path connecting a set of cities all interconnected with direct roads (as in figure 1).

Given n cities, a solution is $x_i = \{c_1, c_2, ..., c_n\}$ and the solution space S has size $n!$.

For this reason it's often not feasible to completely explore S and it's only possible to find an acceptable solution through an heuristic algorithm.

2 Ant System

The Ant System (**AS**) is a metaheurstic introduced in the 1992 by Dorigo, inspired by Goss and Dereborg experience about the ant behaviour, to optimize combinatorial problems.

The **AS** is a swarm intelligence (or collective/group intelligence) method, where many simple individuals are used to explore the research space S using the knowledge of all the group. This means that the collective intelligence and its perception are far better than the ones of the single individuals.

Gros and later Deneubourg studied how ants are able to find the shortest path to the nutriment source in a controlled environment, as the one shown in figure 2.

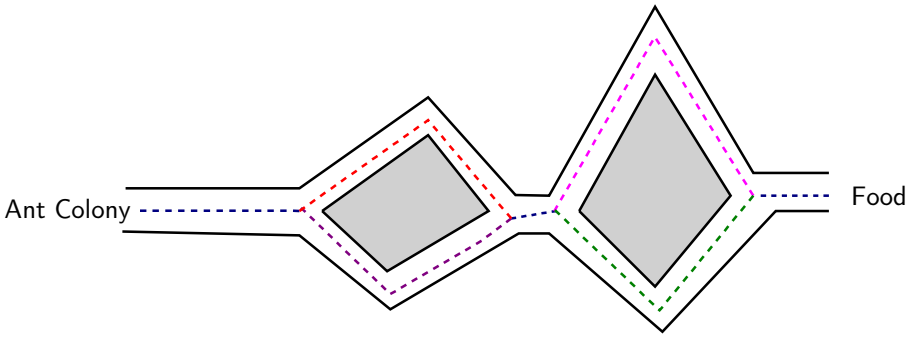


Figure 2. Example of environment used in the Gross experience

This environment was built using pipelines with different lenght connecting the ant colony to the food source. The experience shows that a great majority (90%) of the ants after a short period of time was choosing the shortest path.

This happens because every time an ant is moving it releases a chemical called *pherormone*, and naturally the ants follow the strongest path of pherormones. As the rate of the movements on the shortest path is higher, since the time needed to cover it is smaller, the trail of pherormones will be stronger.

Another property of the pherormone is that it naturally evaporates. This implies that the pherormone left in the longer and less used path will at some point disappear.

Gross and Deneubourg proposed the following mathematical modelling of the ant behaviour at a 2-way crossing:

Given

- L_m: the number of ants that already passed on the L (lower) path
- U_m: the number of ants that already passed on the U (upper) path
- m: the total number of ants that already passed $(L_m + U_m)$

The probability P_U that the $(m+1)$th ant will choose the U path will be

$$P_U(m+1) = \frac{(U_m + k)^h}{(U_m + k)^h + (L_m + k)^h}$$

And the probability P_L that the $(m+1)$th ant will choose the L path will be

$$P_L(m+1) = 1 - P_U(m+1) = \frac{(L_m + k)^h}{(U_m + k)^h + (L_m + k)^h}$$

Where k and h are parameters representing respectively the minimum quantity of pherormones needed to influence the choice and the ant perception of the pherormones.

The results of the interactions between the ants are structures auto-organized in time and in space, where there are no differences of knowledge or importance among the individuals and where none of them has the global knowledge.

This kind of auto-organized systems is very interesting because the individuals are very simple and the sytems are in general very robust with respect to errors and malfunctions, indeed some times the performance can even benefit from possible errors. Moreover these systems are easily parallelizable.

3 The Ant System and the TSP

In our case we will study the adaptation of the **AS** to the **TSP** problem.

As for any other metaheuristic, we have to define some elements:

- The solution space S
- A start solution x_0
- The neighbourhood of a solution $N(x_i)$
- The exploration operator U
- The stop condition

The solution space S will be the one already defined in the section 1 for the **TSP**.

The start solution x_0 is simply a random solution extracted from S.

The neighbourhood of a solution $N(x_i)$ is the entire solution space S, but it will be not explored, instead the successor x_{i+1} will be created by the exploration operator U.

As the **AS** is based on the collective knowledge of m ants at every iteration the algorithm will explore m solutions.

The exploration operator $U(t)$ of an ant will create the successor solution from the path of the ant itself.

Given $\eta_{(i,j)}$ the visibility (the inverse of the distance) of the city j from the city i and $\tau_{(i,j)}(t)$ the pherormones accumulated in the path between the city j and i at the time t, the ant will choose the next city with probability

$$P_{(i,j)}(t) = \begin{cases} 0 & \text{if } j \notin J \\ \dfrac{[\tau_{(i,j)}(t-1)]^{\alpha}[\eta_{(i,j)}]^{\beta}}{\sum_{l \in J} \{[\tau_{(i,l)}(t-1)]^{\alpha}[\eta_{(i,l)}]^{\beta}\}} & \text{if } j \in J \end{cases}$$

where J is the set of all the cities not yet visited by the ant.

α and β are parameters, a bigger α will favour the pherormone trails and so the intensification while a bigger β will favour the visibility and so the diversification.

It's very important to note that at a time t the m each ant will move indipendently from the others, and it is influenced only by the pherormone trails leaved at the time $t-1$.

For this reason the pherormone trails are updated at the end of the iteration t with the following formula:

$$\forall (i,j) \in S \quad \tau_{(i,j)}(t) = (1-\rho)\tau_{(i,j)}(t-1) + \Delta\tau_{(i,j)}(t)$$

with $\Delta\tau_{(i,j)}(t)$ defined as:

$$\Delta\tau_{(i,j)}(t) = \sum_{k=0}^{m} \Delta\tau^k_{(i,j)}(t)$$

and $\Delta\tau^k_{(i,j)}(t)$ of the kth ant as:

$$\Delta\tau^k_{(i,j)}(t) = \begin{cases} 0 & \text{se } (i,j) \notin T^k(t) \\ \dfrac{Q}{L^k(t)} & \text{se } (i,j) \in T^k(t) \end{cases}$$

Q is a constant, $L^k(t)$ the total length of the itinerary found at the time t and $T^k(t)$ the set of the itinerary.

The end condition is simply a maximum number of iteration $t_{\max}$.

4 Implementation

Once again I implemented the algorithm in C++ using extensively the object-oriented features of the language, reusing as much code as possible from the previous TP.

I choose to code the problem using 3 different classes.

The first class represents the collective knowledge, that manages the τ, $\Delta\tau$, η and probability matrices, all $n \times n$ and simmetric.

The visibility η matrix is computed once for all, as the distance between the cities does never change.

The τ and probability matrices are updated at the end of every iteration while $\Delta\tau$ is updated every time an ant ends its walk.

The probability matrix contains the value of $[\tau_{(i,j)}(t-1)]^\alpha[\eta_{(i,j)}]^\beta$ for each pair of cities. This permits to avoid to compute it for every ant and every city combination, but unluckly there is still the need to normalize such number.

The second class represents a single ant, and implements most of the logic presented before.

Finally the ant system class contains the m ants and executes the sequence of operations.

For the α, β and ρ parameters, I used the given values, respectively 1, 5 and 0.1.

For Q and τ_0 (the initial pherormone value), I used L_{nn} and $\frac{1}{L_{nn}}$, where L_{nn} is the lenght of a solution of a greedy heuristic. I presented the greedy heuristic in the previous report.

Finally I will discuss in the next section the values of m and $t_{\max}$.

5 Results

This time I will skip the analysis of the landscape as I analyzed the **TSP** landscape in the previous TP.

However before comparing and analyzing the performance of my **AS** implementation I made some experiments on how m and $t_{\max}$ impact the quality of the solution and the computational performance of the algorithm.

To do so I used the file *cities.dat*, of whom I discovered the global optimal solution in the last TP, that has length of 27.5154.

Using a simple bash script, I run the code with different configurations of m and $t_{\max}$, the results are summarized in the plots of figure 3.

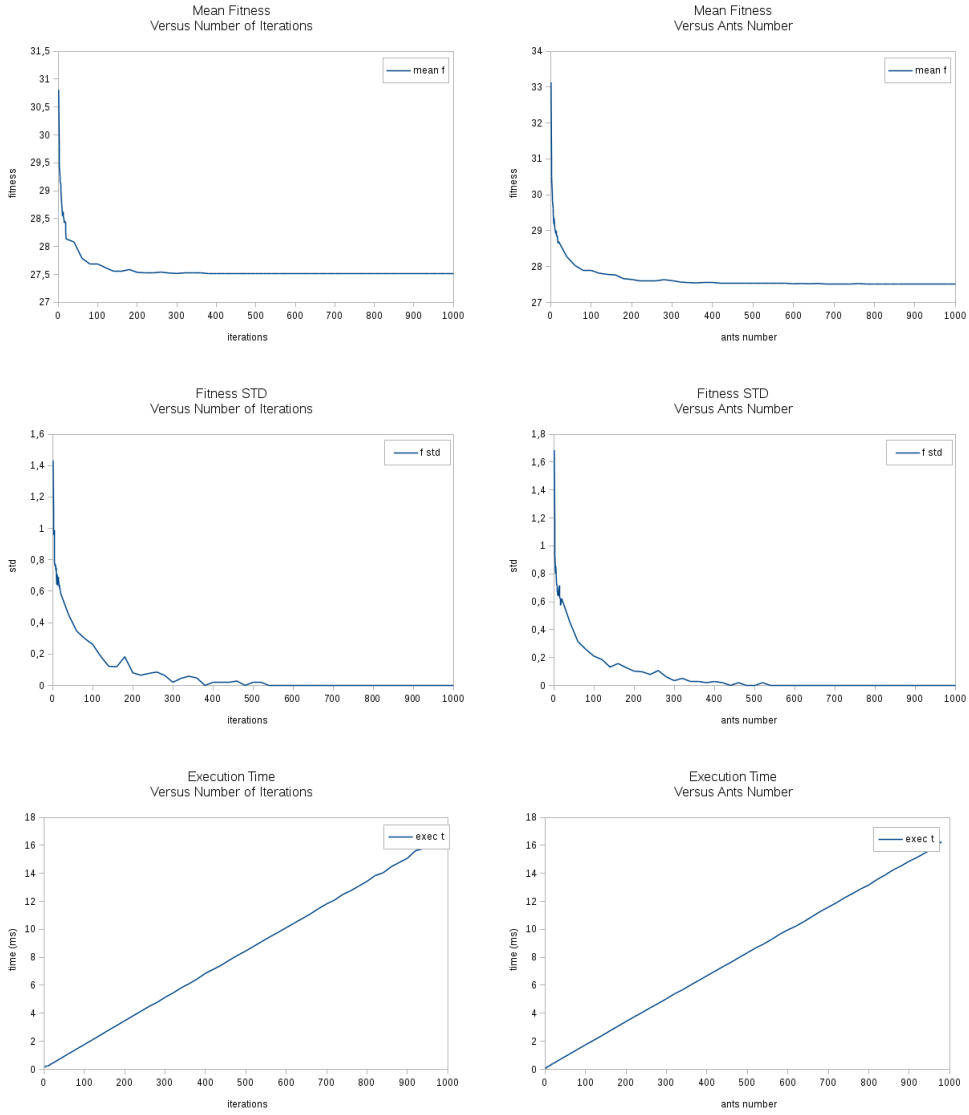


Figure 3. Mean fitness, std and execution times varying $t_{\max}$ and m

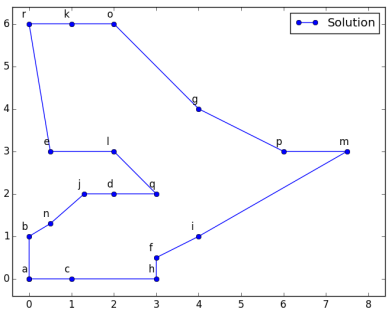
The left figures represent the variations of the mean fitness, the standard distribution and the execution times of 100 execution of the algorithm with $m = 10$ ants, depending on the value of $t_{\max}$ (varied from 1 up to 1000).

The right figures represent the variations of the same parameters with $t_{\max} = 10$ depending on the value of m (varied from 1 up to 1000).

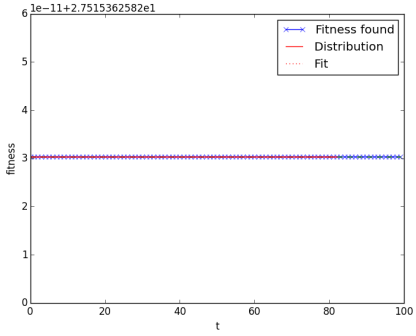
The impact of the parameters m and $t_{\max}$ is equal, and in both cases the execution time has a linear trend while the solution quality improves much faster and converges around 540 iterations and 10 ants or 540 ants and 10 iterations.

For the next experiments, I choose to use a combination of $m = 160$ ants and a $t_{\max} = 30$, the code executed 100 times over the file *cities.dat* gives a distribution with standard deviation 0 and execution time of around 9 ms.

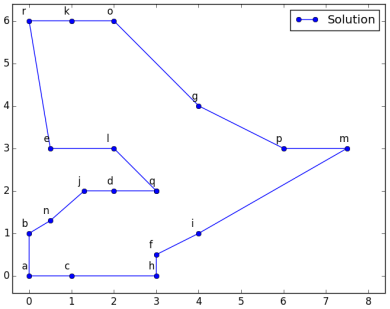
I was asked to compare the results of my **AS** algorithm with the **Greedy** and the simulated annealing (**SA**) metaheuristics, both implemented for the previous TP, over 10 runs. However I chose to run it 100 times to have more relevant data. The results for *cities.dat* were the following:



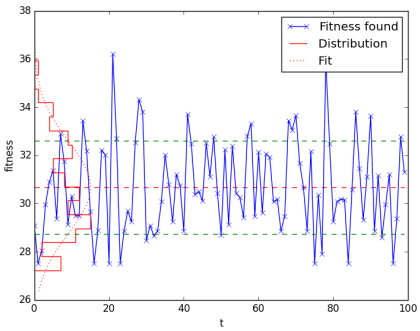
AS best solution



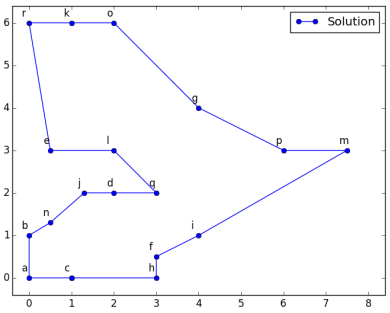
AS distribution



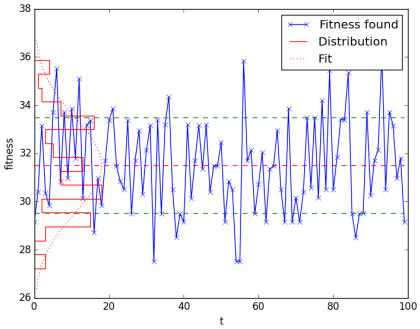
SA best soltution



SA distribution



Greedy best solution



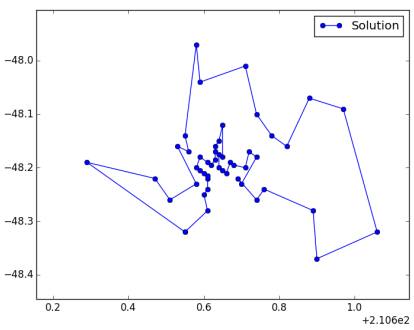
Greedy distribution

Figure 4. Results of 100 runs over *cities.dat*

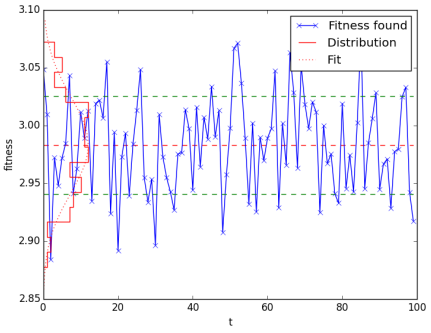
As predictable (knowing the results of the previous TP) the three algorithms are capable to find the optimum solution, however the **AS** outperforms the other two, finding the optimum in the 100% of the cases, while the **SA** in the 7% and the **Greedy** only in the 3% of the cases.

The **AS** is performing very well, with an execution time of 8.0 ms per run, while the **SA** needs 15.6 ms and the **Greedy** needs only 0.1 ms.

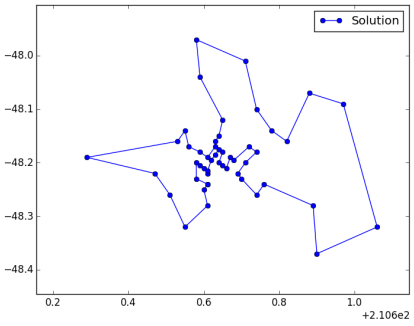
I made similar comparisons for the file *cities2.dat* and the results were the following:



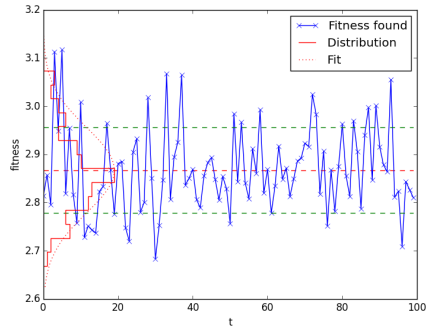
AS best solution



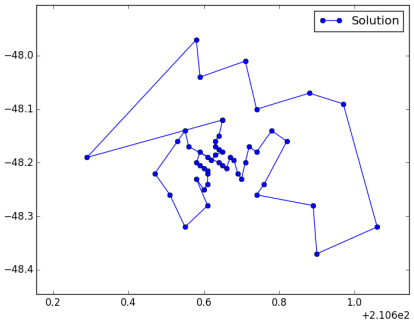
AS distribution



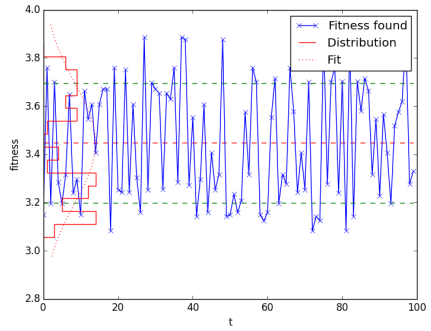
SA best soltution



SA distribution



Greedy best solution



Greedy distribution

Figure 5. Results of 100 runs over *cities2.dat*

In this case the **SA** is performing better then the **AS** and the **Greedy**. The best solutions of the algorithms, as shown in figure 5, are different. For the **SA** the best solution has a length of 2.68 while the one found by the **AS** algorithm has a length of 2.88 and finally the one found by the **Greedy** has length of 3.08.

However the distributions of the **SA** and of the **AS** are similar, the first with average of 2.86 and standard deviation of 0.09 and the second with average of 2.98 and standard deviation of 0.04, while the **Greedy** has an average of 3.44 and standard deviation of 0.25.

Even the difference of execution time between the **AS** and the **SA** is smaller, with the first that needs only 42.3 ms and the second that needs 75.8 ms, with a ratio of 1.8, while with the file *cities.dat* the ratio was close to 2.0. Again the **Greedy** algorithm has an execution time far better of only 0.8 ms.

However I belive that the quality performance could be improved tuning the parameters of the **AS** to increase the diversification to explore new zones of the solution space S.

The following table will summarise the results of the previous experiments, plus the ones made over random problems of size 50, 60, 80 and 100 (always over 100 runs):

File	n	Algorithm	Best length	Average length	STD	Average test time
cities.dat	18	AS	27.52	27.52	0.00	8.0 ms
		SA	27.52	28.35	0.67	15.6 ms
		Greedy	27.52	31.79	2.17	0.1 ms
cities2.dat	49	AS	2.88	2.98	0.04	45.2 ms
		SA	2.68	2.86	0.09	75.8 ms
		Greedy	3.08	3.44	0.25	0.8 ms
cities50.dat	50	AS	74.82	77.54	1.23	49.0 ms
		SA	72.58	82.51	4.01	118.3 ms
		Greedy	78.09	87.03	5.17	0.7 ms
cities60.dat	60	AS	108.01	115.68	2.13	68.0 ms
		SA	109.15	119.55	4.71	135.0 ms
		Greedy	120.68	127.01	4.46	0.8 ms
cities80.dat	80	AS	147.05	152.48	2.02	109.1 ms
		SA	147.86	162.09	6.48	161.7 ms
		Greedy	144.74	157.56	7.97	1.1 ms
cities100.dat	100	AS	72.59	76.25	1.31	161.9 ms
		SA	77.46	90.79	5.89	159.5 ms
		Greedy	72.60	83.00	5.12	2.0 ms

Globally the **AS** has good quality performance, with comparable (or even better) results than the **SA** metaheuristic and it is in average faster than the **SA**. However, as the **AS** stop condition is a maximum number of iteration, the execution time grows quadratically with the complexity of the problem. The **SA** has a more complex stop condition that makes the execution time much more variable and dependent on the landscape of S.

NB: I executed the codes over a different (and more powerful) machine, and the time reported in the table is the average execution time (not the total execution time like in the last TP).

6 Project structure

compile.sh	is the compilation script
compile_basic.sh	is the compilation script without plot dependencies
src/	contains all the source codes (above the core codes)
→/tsp.*	is the code that defines the TSP solution, neighbours, etc
→/antsystem.*	is the code that defines the AS metaheursitic
→/annealing.*	is the code that defines the SA metaheuristic
→/greedy.*	is the code that defines the Greedy heuristic
bin/	contains all the binary, dat files and the compilation script
→/antsystem	runs the AS over a specified file
→/annealing	runs the SA over a specified file
→/greedy	runs the Greedy over a specified file
→/antsystem_ants.sh	script used to analyse the influence of m on the AS performance
→/antsystem_iters.sh	script used to analyse the influence of $t_{\max}$ on the AS performance
bin/csv	contains the csv generated by my scripts

All the executables if executed without arguments will show the basic usage.

The codes and the reports can also be found on my github repository: Metaheuristic.