Tommaso Ravasio, Bemacs 25
ID: 3192281

COMPUTER PROGRAMMING INDIVIDUAL PROJECT

Introduction

In the zip file is contained this PDF file with the answers to the questions, the generate_data.py file and the Individual_project.py which contains the code relatives to each answer.

Question 1

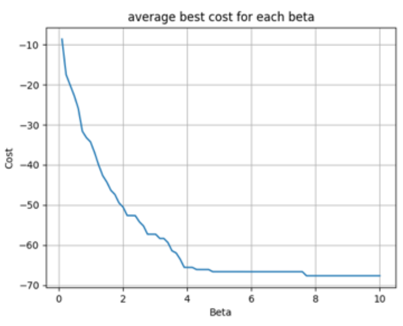
Code in the Individual_project.py file

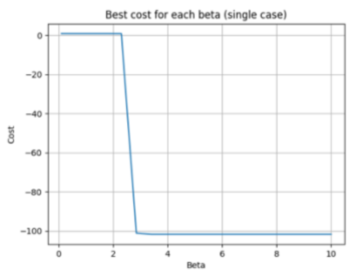
Question 2

It is more convenient from an implementation point of view as if we did not want to allow border jumping we would get an index error when reaching an edge and proposing a move out of the matrix.
From a mathematical point of view instead it is convenient to allow borders jumping as if not the simulated annealing would not respect the detailed balance condition which impose that the probability flows are balanced. To see this let's simplify our problem and reason in 1D terms.
If this condition is not respected the probability of getting a border point from a non-border one is 1/2. Conversely, if we are not allowing border jumping, when moving from a border we can only move to the non border (example: if we are in left border we can only go right) hence the probability of moving to the non border is 1 as we cannot do anything else.
In order to not allow border jumping we should either change the acceptance probability to A_{ij}=min(1,$C_{ji}p_i/C_{ij}p_j$). In code terms we should change the accept function and check with an if condition if the point is a border point, if yes use this formula. We should also change the propose_move to behave differently when we are in a border point.
Otherwise we could add a marginal layer of np.inf to the matrix so that there is the possibility for border point (actually ex border point) to move both ways. But since the cost of the external layer is np.inf, they will always choose to move in the "internal" direction.

Question 3

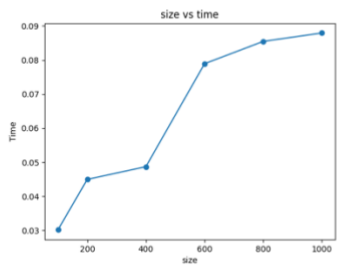
The first graph shows us that on average the cost decreases as the betas increase. It shows also that this drop happens for the first values of beta. This hypothesis is confirmed by plotting the graph for the single case which shows us that often, at a certain point (usually for the first values of beta), there is a drop in the cost and after this drop the cost is stable. We provided here an example. This hints us that probably at a certain step our position finds a value of C where the cost significantly drop, as mentioned above. The cost reached after this drop can be around -50 or -100.

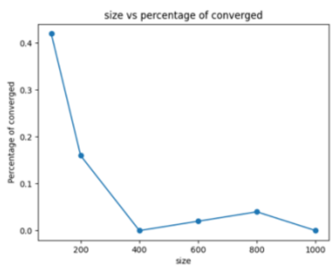




We can hypothesize that there are two areas in the matrix with significantly lower costs. After falling in one of the two “holes” the cost does not change much, hence their bottom is probably almost flat. The annealing process is useful, as otherwise we would get stuck in a local minima during the initial "exploration" phase before being able to fall into a hole and finding a much lower cost.
Code in the Individual_project.py file

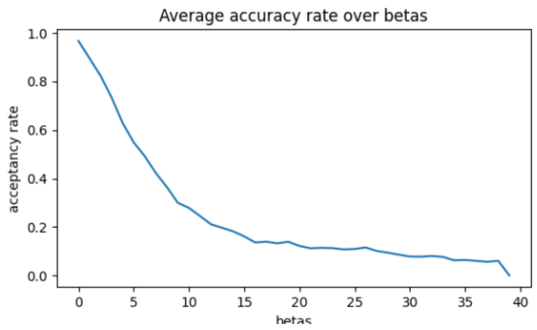
Question 4

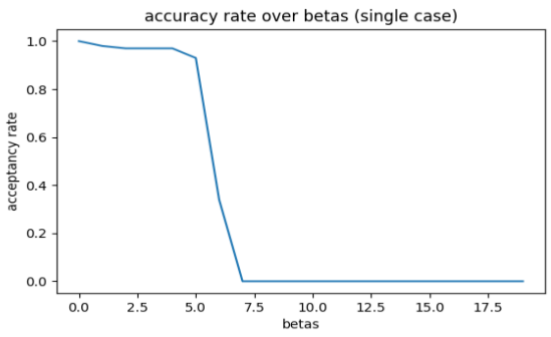




The graphs show that as the size of the matrix increases the time requested to converge increases while the percentage of successful cases that is the percentage of cases in which the simann converged to the solution decreases (we used that the best_c had to be between -105 and -95 as we knew that the minimum was around 100).
This is also intuitive as the bigger the matrix the more time consuming is to fall in the holes above mentioned and also the less probable it is.
Code in the Individual_project.py file

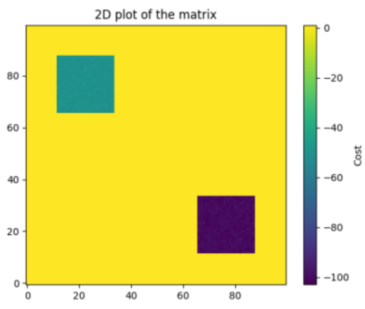
Question 5
As the initial point of the algorithm is random, we iterate the simann 50 times and keep the average of the acceptancy rate recorded during these iterations. By doing this we see that our expectations were correct. That is, as beta increases we see a decrease in the acceptance rate. However, as in point 3, it is very important to note a particular behavior that emerge by looking at the plot of the single case that is that in most of the cases at a certain point there is a drastic drop in the acceptancy rate. This agrees with our previous hypothesis as it means that it fell in one of the two holes and all the moves that would get out of the hole are not accepted.





Code in the Individual_project.py file

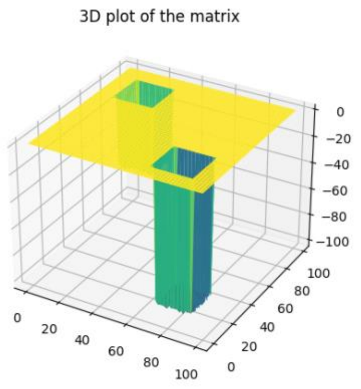
Question 6



Plotting the matrix both in 3D and 2D confirms our thesis formulated in the previous answers.
We can divide the matrix surface in three groups:
1. quasi-flat surface: the majority of the surface of the matrix, its cost is around 0
2. first hole: smaller surfaces where costs do not deviate much from -50
3. second hole: same features of first hole but with cost around -100

The observed drastic drop in the acceptancy rate is due to the fact that our point falls

in one of the two holes, and then it refuses each move that wants to get out of the hole.
It confirms the analysis on the beta values too, as we found that for lower betas we accept almost every move. Therefore we just "explore" our grid, in this phases we are moving on the quasi-flat surfaces and the higher temperature helps us overcome the small changes in cost to keep exploring.



Code in the Individual _project.py file

Question 7

Simulated annealing has the great advantage that it can easily avoid getting stuck in local minima, which represents a problem for other optimization algorithms, such as the greedy. A greedy strategy, indeed, would get stuck quickly in a local minima before even being able to fall in one of the two holes. However, just running the simulated annealing is not the best option for solving this as we have seen that it performs well finding the global minima in most of the cases for smaller n; as n increases, the convergence rate decrease hence we find less times the global minima. To solve this issue, we can run the simann many times and pick the best cost found among the results of all the iterations; of course this method increase performances but decrease efficiency. Another way could be to allow jumps in random points and not only to near points by picking random x and y. In this way, we would be able also in large n to fall in the holes we are looking for. Additionally, if our move t is in the holes with cost -50 and our move t+1 fall in the holes -100, we would also be able to exit the first hole, which we are not using the traditional simann. Of course, this implementation is useful to use because we already know the shape of the matrix, otherwise, generally it's not a better performing algorithm than the simann.