Name: Eleftherios Toramanidis

Student number: s1958459

Prototype Algorithm for TSP

As the title above suggests, the way I have chosen to tackle this problem is better used on the metric TSP. This will become more obvious once I explain how the algorithm works. At this point I would like to make clear that I did not see the idea from someone else, so it is expected to have a lot of flaws. The idea comes from a one-dimensional graph where we again try to find the order to visit all the nodes. The solution is obvious for the one-dimensional case, but that is where I got my idea from and in the future, I am interested in devoting more time to transfer such a solution to 2 dimensions (even 3). Looking at our one-dimensional case, say we have 5 cities that we need to visit, and they all belong to the x axes. The very obvious solution is to take them with increasing order. So, if the cities that we had to visit were sitting on the points 3,6,7,2,1 on the x-axes then the best order to visit all the cities would be to start from 1->2->3->6->7. Note how the starting point does not actually matter. In the same circle described, we can start from 3 and go to 6 etc. Now it seems easy to say that in this case of 1 dimension we simply arrange them with increasing order, but let us suppose that we name the cities a, b,c,d,e. If we make things more abstract, we will need to find another relation that will find the correct order. The observation we make from the first example 1->2->3… is that the outer points, which in our case are 1 and 7, have a higher average. The word "average" in this report will be used as the average of all the transition from our starting point to all the others. For instance, the average of 1 is (1->2 + 1->3 + 1->6 +1->7) / total number of transitions (=4). That yields an average of (1+2+5+6)/4 = 14/4. Respectively the average of 2 = (1+1+4+5)/4 = 11/4. The averages of 7 and 1 are the worst averages and that is why they are placed on the outer parts of the circle. Now that is simply an observation for the one-dimensional case. After realising that fact, I decided to try the same approach even on two dimensions, but unfortunately that does not yield a good tour Value every time. There are cases that the Greedy algorithm works better than my approach, but I still believe that the original idea of taking those "averages" can be useful, if I modify it to work for two dimensions. I would also like to outline, that my approach will for sure not be a good approach for non-metric TSP. In two dimensions, by calculating the averages, we can still find the points that are outer in our 2-dimensional graph (if we were to place the points on a graph with x and y coordinates). I believe that the algorithm will still find the worst averages and place them on them on the outer parts of our circle and as we go inner in our circle(tour) the lower the averages get (which also means that the points will be closer to the centre of the whole graph). The algorithm that I have found, will firstly work the averages of all points from 0 until (n-1) and right them in an array called averages. Then we take the highest average in the array, we find the index and we put the index at the beginning of a new array called x. We basically sort the index with decreasing order (decreasing averages) in the array x. After that the first two elements of x are placed on the edges of our circle (as they have the largest averages) and then we start placing the other points depending on the averages. The first two points are placed as I explained before, but from now on there is another factor we need to consider. For the next points we have already organised them depending on the averages, but to place them in our circle we need to see to which edge are they closer to and place them accordingly. For instance, suppose in the one-dimensional case we have already found that on the outer parts of our tour, we have found the points 1 and 7 on the left and right hand side

respectively. Then it is the turn of 2 to be placed somewhere and then 6(average of 2=11/4, average of 6 = 13/4). To see where we place number 2 we have to compare the distances from 2 to the left hand side and right hand side. We can see that point 2 is closer to 1 so it is placed after 1. At this current stage our circle is 1->2 ….->7. We do the same for 6 which has the next largest average in our x array, and we see it is closer to 7 so our circle becomes 1->2….->6->7. We do that with all of the points until we have reached the end of array x.

Let us check the running time of this algorithm. Firstly we iterate through each point and we calculate all the distance starting from this point. So, for all n points we do n-1 additions to find the total distances and we divide to find the average. That takes n^2-n time, which yields Θ(n^2) so far. In my code, it is more obvious how the time for averages is n^2 since I have a for loop inside a for loop which both run for n times. Then there is another for loop that takes the elements from the array x and places them in the desired order I explained before and that for loop runs for n times. In total we can safely say that the prototype algorithm runs in Θ(n) time.

Experiments

The idea of this python file is that it asks the user, how many points to they want to create and what type of graph they want to use to represent the points (either Euclid or Metric weighted values). The code generates random values and creates a file. For the Metric case the file created is randomeExamples.txt and for the Euclid case we create the randomExamplesEuclid.txt file. Then, we create a Graph, and we calculate the tour Value upon creation. Then we use our prototype algorithm and recalculate the tour Value. If the value, we calculated after using our prototype algorithm is less than the starting tour value, we give the True value to the array. I chose to take 100 examples (all of the same size) to check how the code operates at 100 different occasions (of the same number of points). The code prints the improvements compared to the Greedy algorithm. I did run a lot of examples and apparently my code seems to not be better than the Greedy algorithm, in a lot of cases. I did expect such a thing could happen, but I believe that the algorithm can be improved to be better. When compared to the initial ordering (upon creation), the algorithm improves the tour value almost always.