ENGRI 1101 Engineering Applications of OR Fall 2020 Lab 1

The Traveling Salesman Problem

Name: _____

Objectives

- Introduce students to a real world problem solved by OR practitioners
- Demonstrate the use of heuristics to obtain good solutions to optimization problems
- Give students an appreciation of the difficulty of solving optimization problems exactly

Optional Reading

- Read Handout 2 on the traveling salesman problem.

Brief description

Finding an optimal solution to a Traveling Salesman Problem, and proving that it is, in fact, an optimal solution, is a difficult task. In practice, when a feasible solution to a difficult problem needs to be provided quickly, one often resorts to using **heuristics**, i.e., procedures for generating feasible solutions, or improving existing ones, that can be executed quickly and, hopefully, produce a pretty good result. In this lab we will consider several such heuristic procedures for the TSP.

Here are some of the heuristics we will consider for constructing a solution from scratch:

Nearest neighbor: This is, perhaps the simplest procedure, which works as follows: start the tour at an arbitrary city (or point, or star, or DNA string. From now on we will call it "city" or "node".) At each step, choose the next city to go to by picking the city closest to your current location (among the cities you haven't already visited). Once all the cities have been visited, return to the first city.

Nearest insertion Start with a "tour" on two of the cities (e.g., the closest pair of cities). Find new city closest to any city currently in tour. Insert the city into the tour at the best place.

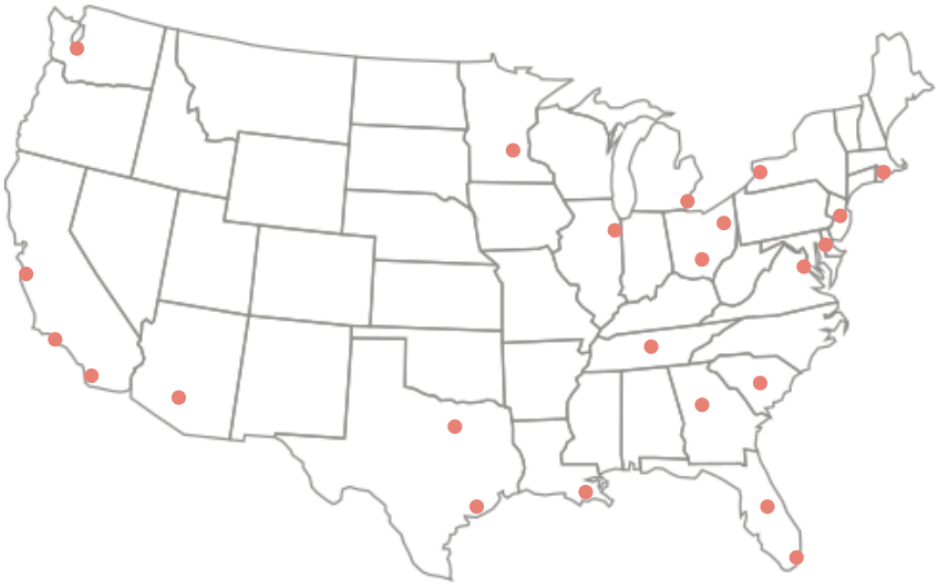
Farthest insertion This algorithm is similar to the previous one: except the new city to insert is chosen as follows: for each city not already in the tour, it computes the distance to the closest node already in the tour, and selects the node for which this distance is **maximum** — may seem counterintuitive, but often works very well in practice!

Once we have constructed a tour, we can try making some small changes to make the tour better. For instance, let us pick two edges in the current tour. Suppose we remove these edges, and add two different edges to "reconnect" the tour. (Notice that there is only one way to do that!) If this change improves the cost, make the change, and repeat the process on other pairs of edges, until no improvement can be made. (This only works for symmetric TSPs — do you see why?)

The above improvement procedure is called 2-OPT. We could get more sophisticated — removing 3 edges at a time, for example, and reconnecting the tour (this is called 3-OPT). There are even more sophisticated (fast) schemes. Notably, the Lin-Kernighan heuristics, which won't be covered, but can easily be googled.

Part #1: Solving the Problem Manually

To get an appreciation for the complexity of the problem, first go to `http://engri1101.orie.cornell.edu/`. Click on the `Traveling Salesman Problem` words to bring up the following image.

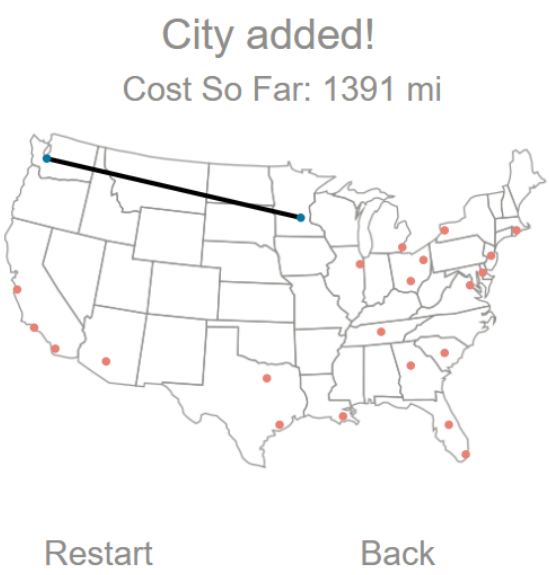


This image shows the 23 locations where Beyoncé performed in the U.S. on her *On the Run II Tour*. Many factors would have determined where Beyoncé performed on what dates, and in turn, the order in which she visited those locations. Let's suppose, though, that Beyoncé's primary concern was spending as much time performing and making music as possible. To that end, she might want to visit the 23 concert venues in the order that minimizes the total distance travelled.

Try solving the TSP for this instance. Even with 23 cities, it gets tricky! First, draw a solution by hand on the above map.

Now lets compute the total distance your route requires:

- Go back to the map on `http://engri1101.orie.cornell.edu/`.
- Click any node as the 'first location' of your tour. From there, click the next city on your route. For example, if your route has Beyoncé go from Seattle, WA to Minneapolis, MN, you could first click Seattle and then Minneapolis. You should see the first arc of the tour created:



- Continue successively adding cities, one at a time, in the order you drew. If you make a mistake, hit `Back` to remove the city you most recently added. You can also click `Restart` to restart the entire process.
- when you click the final 23rd location, it will automatically be connected to the first, creating a tour. Note that, if any node is still pink, that means it has not been clicked.

What is the total distance your tour requires Beyoncé to travel? This is reported above the map!

Part #2: Heuristics for constructing tours from scratch

In the pre-lab, you considered one particular input for the traveling salesman problem. In the first part of the lab, you will get a feeling for how a few different heuristics work, by running them on the data set that you solved by hand, as well as a couple similar instances.

You will be asked to try the first three algorithms: nearest neighbor, farthest insertion, and nearest insertion.

Part 2#a: The 6 by 8 grid graph

Back on `http://engri1101.orie.cornell.edu/`, click the button that says `(Next) Part 2a: 6 × 8 Grid`. You should be brought to a 6 by 8 grid graph and will be able to select from several Heuristics.

Originally, `Mode` option should be set to Manhattan. This means that distances will be sum of vertical and horizontal distances[1]. You can then click any of the four heuristic options listed: the latter three were explained

[1]This is what happened in the prelab example: there is a route between any two nodes ("cities"), and the cost of that route is the sum of the vertical and horizontal ditsances separating them. E.g. if you have to go 4 to the right and 4 up to get from node a to node b, then the cost is $c[a,b] = 3 + 4 = 7$. The name "Manhattan" comes from thinking of cities with roads that form a grid.

at the beginning of the lab, while the first (`Random Neighbor`) generates a completely random tour (this is a terrible method, but useful as a baseline comparison).

First, click on `Nearest Neighbor`. Press 'd' as prompted to watch the nearest neighbor algorithm build the tour. Although you can blithely just hit this key over and over again, **DO NOT DO SO**. For each move, try to figure out what the algorithm does next, and then double check by typing 'd'. What sort of "mistakes" does the algorithm make? Note: sometimes a new edge is not easily visible, because it lies almost completely on top of several edges already selected for the tour.

Once your path has exactly 3 nodes (and 2 edges), hit `Back` once, then hit 'd' once (if you have already added more than 3 nodes, hit `Restart` and restart running the heuristic until you have 3 nodes). Repeat the `Back` + 'd' process several times. You should notice that the next node added is not always the same – but that it is always the same distance. Why do you think this is? What do you think this means will happen if you run the Nearest Neighbor heuristic in full several times?

Now run the algorithm in full. Write down the value of the solution you got.

Try it again. That is, click `Restart` and hit 'd' to run the algorithm in full. Does it compute the same tour? Why do you think this is?

Now try running the nearest insertion algorithm: it chooses the node closest to any node already in the tour, and then inserts it into the tour at the best place. More specifically, click `Nearest Insertion` on the screen, then click 'd' repeatedly. Again, try to predict and understand how the algorithm is updating the tour in each iteration. How does this algorithm compare? Try running two or three times.

Another way to construct a tour is the farthest insertion algorithm. At each step, for each node not already in the tour, it computes the distance to the closest node already in the tour. It then selects the node for which this distance is maximum, and inserts this node into the tour in the best place. Click `Furthest Insertion` on the screen, then click 'd' repeatedly. How does it compare? Can you give any intuition why farthest insertion works

well in practice?

Compute 3 random tours using the `Random Neighbor` button. How does the best of these compare to the best one you computed above? Can you give some explanation for this?

Part 2#b: The 9 by 9 grid graph

Now try running all these algorithms with the larger graph grid99. To do so, click `(Next) Part 2b: 9 × 9 Grid`. This time, however, make sure that the `Euclidean Distance`[2] mode is selected. Does the comparison between the heuristics change?

[2]The Euclidean distance between two points is their ordinary distance. If you have to travel right 3 nodes and down 4 nodes, then the Euclidean distance is $\sqrt{3^2 + 4^2} = 5$.

Part #3: Heuristics for tour improvement

Now we will try out the 2-OPT improvement algorithm. Click `(Next) Part 3: 2-Opt`. Follow the instructions on the web app. You will be guided to start from a random tour and run 2-OPT one step at a time mode. Do it several times. Did you get different results?

Try to explain why or why not. For any data set with Euclidean distances, is it possible that an optimal tour crosses itself? Explain why or why not. (In other words, either explain why, for any input, the optimal tour does not cross itself, or else, give an input which has an optimal tour that does cross itself.)