02-713 Homework #5: Traveling Salesman and A*
Due: Mar. 18 by 9:30am

Introduction. Let $V = \{u_1, \dots, u_n\}$ be a set of cities and let $d(u_i, u_j) = d(u_j, d_i)$ be the (symmetric) distance between city u_i and u_j. The optimal traveling salesman tour of the cities is a sequence of cities that visits every city in V exactly once, that returns to the city at which the sequence started, and that minimizes the total distance traveled. The TSP problem is the computational problem of finding such an optimal tour. The TSP problem is NP-complete, which we will see later means that it's unlikely that an efficient algorithm exists for this problem. In this assignment, you will implement heuristic search algorithms to find the optimal tour.

Assignment. Write a Python program that finds the optimal traveling salesman tour. You can use any Python library that is part of the standard Python distribution[1]. You can also use the `networkx`[2], `numpy`[3], `matplotlib`[4] libraries. You can also use any code that is on the 02-713 website. You can use any algorithm you want to find the optimal TSP tour; one suggestion is given below.

Input will be provided as a graph in GEXF format, which can be read using the `networkx.read_gexf()` function. The graph is undirected and guaranteed to be *complete* (there's an edge between every pair of nodes). Each node will have attributes '`x`' and '`y`' that are floating point coordinates in the plane for that city. Every edge will have an attribute '`weight`' that gives the length of the edge, which will have been computed by the Euclidean distance:

$$d(u_i, u_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}. \tag{1}$$

See the NetworkX documentation for more details about how to access graph attributes.

Output should consist of two lines of the following format:

```
Tour: a b c d e f g h i j
Cost: 10.2345
```

These lines should start in column 0 and be printed to standard output (using, e.g., the Python `print` statement). The first line gives the optimal tour that you compute. The tour should start with the node that has the lexicographically first name. The node names then should be listed in the order of the optimal tour, each separated by a single space. The first city should not be listed twice. The second line gives the cost of the tour reported on the previous line. Your program can print other lines of output, but only the two above should start with `Tour:` and `Cost:` . Zero points will be give for assignments that fail to format the output correctly.

Command line and runtime environment: You should submit your assignment via Autolab, including all the files need for it to run. You can assume that the computer we use to run your program has `networkx`, `numpy`, and `matplotlib` installed, so you do not need to submit those libraries with your program. Your program should run using Python 2.7. It should be callable via the command line using the following syntax:

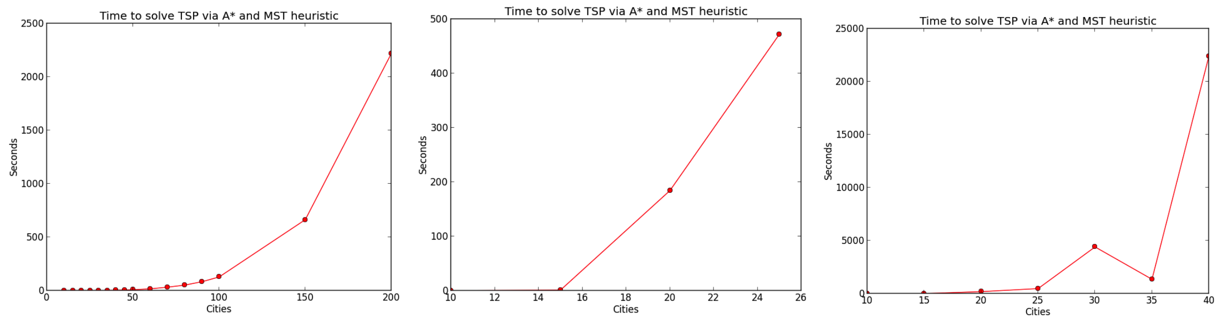
[1] `http://docs.python.org/2/library/index.html`
[2] `http://networkx.github.com`
[3] `http://www.numpy.org`
[4] `http://matplotlib.org`

```
python tsp.py input.gexf
```

where `input.gexf` is the GEXF file to read that defines the cities and distances between them.

Collaboration policy. You must code your own solution to the problem without using or looking at other students' code. You should not use any code you find online (except "reasonable", small, $<$ 5-line snippets that describe how to solve general programming tasks). You can use any other sources, books, websites to design your algorithm. You should cite any source you use.

One approach. You are free to solve the problem using any algorithm you like. A good approach would be to implement the A* algorithm and use it to search a state space graph where the nodes represent prefixes of tours and prefix $a_1, \ldots, a_k$ has neighbors $a_1, \ldots, a_k, a_{k+1}$ for every a_{k+1} that is not in $\{a_1, \ldots, a_k\}$. A good heuristic is the cost of a minimum spanning tree connecting the remaining nodes. Figure 1 below shows the running time on an iMac of this approach for various problem sizes. You code should run on these instances (provided on the website) in approximately the same amount of time or better.



(a) (b) (c)

Figure 1: (a) TSP using A* and the MST heuristic on graphs generated from randomly placed cities where n edges forming a tour have been artificially given weight 0. The instances are given in the `rNem1.gexf` files. (b) Running time for the same algorithm when there is no embedded tour of length 0. The instances are given in the `rN.gexf` files. (c) Three more larger problems added to figure (b). Note the very long time it takes to solve a TSP instance of size 40.