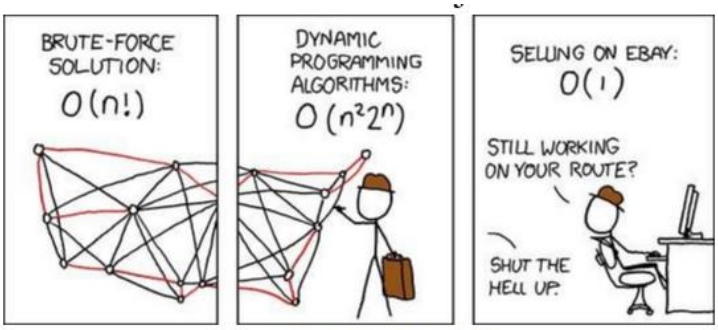
CS 325 Project: The Travelling Salesman Problem (TSP)



from: http://xkcd.com/399

In this project you will have fun trying out ideas to solve a very hard problem: The Traveling Salesman Problem (TSP).

You are given a set of n cities and for each pair of cities c_1 and c_2, the distances between them $d(c_1, c_2)$. Your goal is to find an ordering (called a tour) of the cities so that the distance you travel is minimized. The distance you travel is the sum of the distances from the first city in the ordering to the second city, plus the distance second city to the third city, and so on until you reach the last city, and then adding the distance from the last city to the first city. For example if the cities are Seattle, Portland, Corvallis and Boise. The total distance traveled visiting the cities in this order is:

d(tour) = d(Seattle,Portland) + d(Portland, Corvallis) + d(Corvallis,Boise) + d(Boise, Seattle)

In this project, you will only need to consider the special case where the cities are locations in a 2D grid (given by their x and y coordinates) and the distance between two cities $c_1 = (x_1, y_1)$ and $c_2 = (x_2, y_2)$ is given by their Euclidean distance. To avoid floating point precision problems in computing the square-root, we will always round the distance to the nearest integer. In other words you will compute the distance between cities c_1 and c_2 as:

$$d(c_1, c_2) = \text{nearestint}\left(\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}\right)$$

For example, if the three cities are given by the coordinates c_1= (0, 0), c_2 = (1, 3), c_3 = (6, 0), then a tour that visits the cities in order c_1 –> c_2 –> c_3 – >c_1 has the distance

$$d(tour) = d(c_1, c_2) + d(c_2, c_3) + d(c_3, c_1)$$

where

$$\begin{aligned} d(c_1, c_2) &= \text{nearestint}\left(\sqrt{(0-1)^2 + (0-3)^2}\right) \\ &= \text{nearestint}\left(\sqrt{(-1)^2 + (-3)^2}\right) \\ &= \text{nearestint}\left(\sqrt{1+9}\right) \\ &= \text{nearestint}\left(\sqrt{10}\right) \\ &= \text{nearestint}(3.1622\ldots) \\ &= 3 \end{aligned}$$

$$d(c_2, c_3) = \text{nearestint}\left(\sqrt{(1-6)^2 + (3-0)^2}\right)$$
$$= \text{nearestint}\left(\sqrt{(-5)^2 + (3)^2}\right)$$
$$= \text{nearestint}\left(\sqrt{25+9}\right)$$
$$= \text{nearestint}\left(\sqrt{34}\right)$$
$$= \text{nearestint}(5.8309\ldots)$$
$$= 6$$

$$d(c_3, c_1) = \text{nearestint}\left(\sqrt{(6-0)^2 + (0-0)^2}\right)$$
$$= \text{nearestint}\left(\sqrt{(6)^2 + (0)^2}\right)$$
$$= \text{nearestint}\left(\sqrt{36+0}\right)$$
$$= \text{nearestint}\left(\sqrt{36}\right)$$
$$= \text{nearestint}(6)$$
$$= 6$$

So that d(tour) = 3 + 6 + 6 = 15.

Project Specification

Your group will research at least three different algorithms for solving the TSP problem. Each group member should research a different algorithm and provide pseudocode for that algorithm even if it is not implemented. There is much literature on methods to "solve" TSP please cite any sources you use. You will design and implement at least one algorithm for finding the best tour you can for the TSP problem. TSP is not a problem for which you will be able to easily find optimal solutions. It is difficult. Your goal is to find the best solution you can in a certain time frame. Use any programming language you want that runs on flip.

Your program must:

- Accept problem instances on the command line.
- Name the output file as the input file's name with .tour appended (for example input **tsp_example_1.txt** will output **tsp_example_1.txt.tour**).
- Compile/Execute correctly and without debugging on the flip server according to specifications and any documentation you provide.

Input specifications:

- A problem instance will always be given to you as a text file.
- Each line defines a city and each line has three numbers separated by white space.
    - The first number is the city identifier
    - The second number is the city's x-coordinate
    - The third number is the city's y-coordinate.

Output specifications:

- You must output your solution into another text file with n+1 lines, where n is the number of cities.
- The first line is the length of the tour your program computes.
- The next n lines should contain the city identifiers in the order they are visited by your tour.
    - Each city must be listed exactly once in this list.
    - This is the certificate for your solution and your solutions will be checked. If they are not valid you will not receive credit for them.

Example instances:

I have provided you with three example instances. They are available on Canvas and are provided according to the input specifications.

- **tsp_example_[*].txt** Input files
- **tsp_example_[*].txt.tour** Example outputs corresponding to these three input cases.

The optimal tour lengths for example instances 1, 2, and 3 are 108159, 2579 and 1573084, respectively. Clearly these do not match the values in the tour files. You should use these values to evaluate your algorithm. For full credit it is required that at least one of your algorithms has the ratio of

(your tour length)/(optimal tour length) <= 1.25, and run in less than 5 minutes.

Note: The 1.25 bound is only required for the specific instances that I give you. I am not requiring proof of a 1.25-approximation algorithm. Some 2-approximation algorithms will satisfy the 1.25 bound on these specific instances.

Testing:

You should test that your output solutions are correct. By "correct" we mean that the distances have been computed correctly not that the solution is optimal. The testing program **tsp-verifier.py** can be used to verify that your solutions are valid tours and that the tour distance is correct. Usage to verify a solution to an instance is: (NOTE: requires TSPAllVisitied.py)

```
python tsp-verifier.py  inputfilename  solutionfilename
```

Project Report:

You will submit a project report containing the following:

- A description and pseudocode of at least three different methods/algorithms for solving the Traveling Salesman Problem. Summarize any other research your group did.
- A discussion on why you selected the algorithm(s) you implemented. Describe the implementation with language specific pseudocode.
- For each algorithm implemented, list your "best" tours for the three example instances and the time it took to obtain these tours. No time limit.

- Your best solution for each test instance (which algorithm achieved the best results). Time limit 5 minutes for at least one algorithm.

Submission:

- All files (including a README), solutions and report must be zipped and submitted to TEACH by Friday March 13th at 11:59pm PST. Only the report will be submitted to Canvas.

Check List

- Does your program correctly compute tour lengths for simple cases?
- Does your program read input files and options from the command line?
- Does your program meet the output specifications?
- Did you check that you produce solutions that verify correctly?
- Did you find solutions to the example instances?
- Did you find solutions to the test instances?
- Does your code compile/run without issue according to your documentation?
- Have you submitted your report to Canvas? In the comment section post the onid username of the person who submitted to TEACH.
- Have you submitted your solutions to the test cases, your source code and README file to TEACH?