CS 577: Introduction to Algorithms **Spring 2015**

Programming Project 4: Traveling Salesperson Tours

Goal: To use minimum spanning trees and bitonic tours to approximate the shortest traveling salesperson tour for a complete graph on points in Euclidean 2-dimensional space.

This assignment is a lot of work. If you can get through it, you will learn a great deal about approximation algorithms, minimum spanning trees and dynamic programming. And, you can generate some nice visual output to display your work.

Homework substitution: This programming project may be substituted for the written portion of Hw 6.

Due Date: 4pm Friday May 7th.

Background. The traveling salesperson problem is the following.

> **Given:** A connected graph with positive edge weights.
>
> **Find:** The minimum length traveling salesperson tour. A traveling salesperson tour is a simple cycle that contains all the nodes of the graph, and the length of the tour is the sum of the weights of the edges in the tour.

The traveling salesperson problem is NP-complete and the best known algorithms use time exponential in the size of the given graph. Further, there aren't good, fast approximation algorithms for the problem. But, the optimal solutions are easier to approximate if we put some restrictions on the input graphs. One well-studied restriction is what's called the *metric TSP*. In this case the weights on the edges of the graph satisfy the triangle inequality. This means that the direct distance from node A to node B (= weight on edge [A,B]) is less than or equal to the distance from A to C plus the distance from C to B for any node C in the graph. In other words, direct paths are always as short as paths that pass through another node.

One way to generate weighted graphs that satisfy the triangle inequality is to generate nodes as co-ordinates in Euclidean space and use the Euclidean distance function as the weight of an edge between any pair of nodes. In this programming assignment you will study approximation algorithms for the TSP on Euclidean graphs in 2-dimensional space. This is still a hard problem to solve optimally.

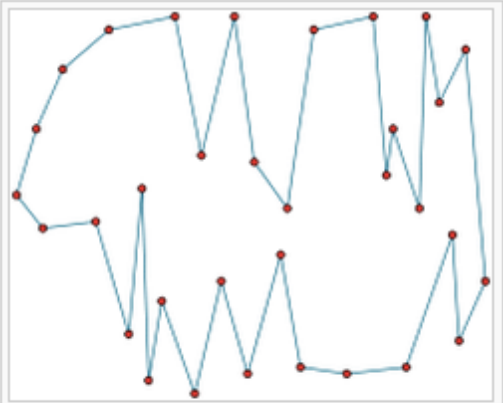
Part 1. Constructing Data Sets. You're going to study two different approximation ideas for the Euclidean TSP. Try to generate test data to show the best and worst of each technique. Since finding the optimal solution is still difficult for Euclidean TSP, you should generate some data sets that are small enough that it's easy to find the optimal tour. Remember that you can find the optimal tour by brute force: just try all possible tours. This certainly works but requires exponential time! Start with small data sets to be certain that your algorithms are working and then build larger test sets.

For the larger data sets you may want to design a data set so that you know exactly what the optimal tour is. For example, all the points might be on a grid, a circle or a polygon. Adding a few "extra" points might "confuse" one approximation algorithm, but not the other. There are also public data sets that serve as benchmarks for people working on TSP algorithms. For some data sets the optimal tour is known, for others not … perhaps some of you can improve on the "best known." I've added a reference below and you can search for more.

Part 2. Approximation Algorithms. The two approximation techniques that you are going to study are: 1) a technique that extends the minimum spanning tree (MST) into a tour and, 2) a dynamic programming algorithm for finding *Bitonic tour*s. The references below give further discussion of these algorithms. But, here are the basic ideas.

For a given graph the MST is the minimum weight sub-graph that connects all the nodes and it's easy (polynomial time and space) to construct the MST. So, it seems reasonable to think we might be able begin with an MST and transform it into a TSP tour. The basic idea is that if we "walked around the outside" of the MST we would get a cycle that visited all the nodes of the graph and returned to the start node. But it wouldn't be a TSP tour because it visits nodes more than once. But suppose on the walk anytime we are going to visit a node already visited, we just skip ahead to the next unvisited node. Doing this transforms the "MST based tour" into a TSP tour. And if the graph is Euclidean, these "skip-ahead short-cuts" are really shorter. Section 9.2.3 of your text discusses this technique. If you think more about this algorithm, you will realize that some "short-cuts" are better than others. An extension of this algorithm (called Christofides Algorithm) uses a technique called *perfect matching* to find the best collection of short cuts. For this assignment you have plenty of work without this extension, but I have included a reference below.

The second approximation technique constructs a tour with a particular geometric property. It's easiest to understand this property by looking at a picture; here's the image used by the Wikipedia article referenced below.



A bitonic tour

If you walked the edges of the tour in this diagram beginning at the leftmost node, you would traverse left-to-right until you reached the rightmost point, then you would traverse back right-to-left to the starting point; hence the name *bitonic*. First sorting the points by x-coordinate and then processing them left to right can construct tours of this type. As each point is processed it is added to either the upper portion of the tour, or the lower portion. A dynamic programming algorithm can be used to make the optimal decision for each point. The reference below gives more details.

Things to study: First, read section 9.2 of your text. In that section there is a discussion of the *approximation ratio* for an algorithm. Basically, this ratio is the value of the approximation algorithm's solution divided by the value of the optimal solution. This is a good metric for evaluating the quality of an approximation algorithm. We would like you to study this ratio for various test sets.

References.

1. From your text: Chapter 6 for the definition of the TSP and section 9.2 for a discussion of approximation algorithms and ratios. Section 9.2.3 of the text discusses the MST based approximation for the Euclidean TSP.
2. Christofides algorithm is an extension of the basic MST approximation. It uses a technique called "perfect matching" to figure out the best way to "short cut" MST tour. http://en.wikipedia.org/wiki/Christofides_algorithm
3. For a discussion of Bitonic Tours for TSP: http://en.wikipedia.org/wiki/Bitonic_tour
4. Data sets: http://www.math.uwaterloo.ca/tsp/world/countries.html

Things to turn-in.

Your Code:

1) A copy of your code and test data. There is a dropbox set up on the Learn@UW site for Programming Project 4. You can simply drop you code there. One copy per group is fine.

A Written Report:

2) Graphs or plots showing the approximation ratio for your tests. Also, plot out your tours so that you can visualize the differences in the approximations.

3) For each graph/plot write two paragraphs: the first explaining the way you conducted the test and the second explaining what you concluded from your tests. This report should be printed and turned in during lecture or placed in the box on the door CS 4382.