# Project 4

CS325 — Spring 2015

---

by Group 2
Vedanth Narayanan
Jonathan Merrill
Tracie Lee

June 6, 2015

## Ideas behind the algorithm

There was a lot of research we did before even coming close to figure out what we wanted to implement. The single algorithm that was universally praised was Christofides' Algorithm. To succinctly explain, it starts out by creating a minimum spanning tree (MST). The next step is trying to figure out the nodes that have an odd number of edges. Now short cuts need to be created, so there are no vertex has any odd number of edges. A Euler path needs to be found now. Without an Euler path, not all edges can be traversed atleast once. This sets us up so we can properly traverse the graph using shortcuts we created, and being able skip over vertices as they seem appropriate. It has been mentioned that Christophides' algorithm is probably one of the best as of now. It is "guaranteed to be at most 50 percent longer than the shortest route," as Erica Klarreich mentions in her Wired article, "Computer Scientists find new shortcuts for infamous Traveling Salesman Problem."

We definitely were not able to implement this algorithm. Our algorithm did incorporate ideas from Christofides' Algorithm though. The Algorithm that we ended up implemented starts off by implementing an MST. But as this is being built, necessary shortcuts are produced. Building the MST happens by comparing the distances from a current vertex and basically all other vertices to find the next closest one. This continually happens until shortcuts need to be created and what not. The final tour starts at the last city and back to the vertex we started at. Because the vertex checks exist, some may call this the brute force method. We know for a fact that since this method is followed, we will get a very accurate answer. It may not take the most optimized path, but it will come close. While Chritofides' algorithm involves some revision, ours just performs the checks it needs to as it builds the tour.

While implementing this algorithm, we drew out little test cases by hand and performed some tests to make sure that it was working it should be. This really helped us out in understanding what we were trying to do. The inputs may just be numbers, but having a visual representation to go along with it helped us with not only understanding what we were trying to implement, but if we were doing a good job of it.

## "Best" Tours

The "best" tours for the three example instances and the time it took to obtain these tours.

## Best Tours for the Competition Instances

Are we taking part in the competition? If we win, we will get extra credit. Winky face.

## References


- http://www.wired.com/2013/01/traveling-salesman-problem/
- https://developers.google.com/optimization/routing/tsp
- http://www.thetubechallenge.com/
- https://github.com/branaugj/cs325/blob/master/proj4/proj4.py
- http://www.cs.princeton.edu/ wayne/cs423/lectures/approx-alg-4up.pdf
- http://www.seas.gwu.edu/ simhaweb/champalg/tsp/tsp.html
- http://web.stanford.edu/ saberi/tsp.pdf


## Check List - Should delete this part in the end

- 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 competition instances? Include a summary of these results.
- Does your code compile/run without issue according to your documentation?