1. Team name: Teamspeed

2. Login: cs170-fr (Arjun Mehta)

3. Names:
Sanket Padmanabhan (24264346),
Arjun Mehta (24488999)
Tom Meng (25638549)

4. We used the Ant Colony Optimization approach for TSP, with some modifications so that it would work with the NPTSP problem. This algorithm is a natural algorithm based on the way ants find food in nature. We start with ants randomly moving about. When one ant finds some food, it leaves a pheromone trail. Ants that move along randomly are more likely to follow a pheromone trail which increases the pheromone trail until all ants follow one specific trail. As time goes on, pheromone trails evaporate which means that ants won't keep going back to old trails when newer, better ones have been found. So Ants (a thread) go around until it finds good paths, and the better the path the more of a pheromone trail it has until the best path is eventually discovered. Our best solutions from part 1 are also hard coded in.

$$\tau_{xy} \leftarrow (1-\rho)\tau_{xy} + \sum_{k} \Delta\tau_{xy}^{k}$$ Pheromone trails are updated by

$$\Delta\tau_{xy}^{k} = \begin{cases} Q/L_k & \text{if ant } k \text{ uses curve } xy \text{ in its tour} \\ 0 & \text{otherwise} \end{cases}$$ and it evaporates in

5. We came up with hard cases on paper, and just wrote python scripts to generate them for us based on patterns we saw within individual cases. We used several strategies to come up with inputs that we thought were tricky, such as isolating all red and blue vertices in 2 separate groups and making cheap cost edges between same colors and expensive edges between different colors.

6. The code has 5 main files.

Solver.py: similar to the python starter code that was given to us. Reads in a file and creates an anttsp to solve it.

Anttsp.py: takes in the number of nodes, the edge weight matrix, and the color matrix. Comes up with how many threads it wants to use to solve this problem and then how many iterations it wants to go before finishing. Its findSolution function will try num_repititions times to create an ant_colony and find the best_path. If it finds a path better than the previous best path it will update it and at the end it will return the best path vector. It also has an ant graph

AntGraph.py: houses all the data structures used for the algorithm. It has delta, which is the edge weights, tau which is the pheremone trails, color which is the color and etha which is 1/delta. It has getter/setter functions for all of these as well as some update and average functions which are useful for the particular formula.

Antcolony.py: creates ants and has an update method that ants can use to update paths and path costs.

It also takes care of pheromone trail evaporation with its global_updating_rule() method.

Ant.py: keeps track of colors and looks for paths. The purpose of ant.py is to represent the behavior of ants. Before we made the change to ant.py, an ant can go to any unvisited nodes with pheromone value greater than average. Basically, what we changed is to constrain the behavior of any ant, so it will not visit three same colored nodes in a row.

First, keep track of the color of previous nodes and the one before that. Make sure the one picking currently doesn't violate the 3 same color restrictions.

Second, keep track of the numbers of each color node was used. The reason for it is that we can't afford to have 1 color run out before the other one, otherwise it is very likely to violate the 3 same color restriction at the end of the path.

Third, when the number of nodes left to visit is less than 5, we made the 2 colors to balance to make sure one color never run out before it is still needed to finish the path. What we did is if number of one color node is less than the other one, just use the other one.

To run the code, simply run "python solver.py" in the code directory.

7. We used the trevlovett/**Python-Ant-Colony-TSP-Solver** from GitHub and formulas above are from the Ant Colony Optimization wiki page.