Solving Travelling Salesman Problem

Ashley Maddix, Camilo Schaser-Hughes, Jason Bockover, Yufei Tao

Assignment Codebase:
https://github.com/camilionaire/AI_GroupProj_TSP.git

Contribution:

Ashley: Worked with team members on the ACO algorithm and project report, and collaborated with members through zoom and slack.

Camilo: Wrote a cheap and easy SA, collaborated with Yufei on the GA and Ashley and Jason on the ACO on concepts, code and tweaking via zoom and slack.

Jason: Ant Colony Optimization algorithm, adding gui menus to main and antColony.py, worked on different parts of project report and collaborated with the team on slack and zoom.

Yufei: Different versions of Genetic algorithm, project report, project's main interface, zoom meeting and slack collaborations.

Introduction:

The travelling salesperson problem is one of the oldest and most studied problems in optimization. In this paper, we are going to analyze and compare three optimization algorithms to find shortests paths for several TSPs. We want to see the efficiency differences between these three algorithms, and analyze each algorithm to see what we can do to optimize the searching efficiency.

Data Collection

For this project, we used four different datasets from TSPLIB. In those four datasets, they contain 5, 26, 42 and 48 cities in each file respectively. The reason we chose these four different numbers of cities is because most of them are evenly spread out. They should correspond to an easy 5 puzzle, France, the famous Dantzig 42 reduction and the contiguous US capitals. They can test how our algorithm performs in different difficulties.

In each dataset, cities' information is represented in an adjacency matrix. The row represents each city and each column represents the distance between the current city and the rest of the cities. These datasets assume each city can travel to any other city on the map.

Approach:

Genetic Algorithm is an efficient approach to these kinds of NP-Hard problems. The genetic algorithm generates a set of potential solutions and refines those solutions by performing crossovers and mutations to narrow down the most optimized solutions. In the Traveling Salesman problem, GA generates a given set of potential routes that contain each city once as the initial population. The chromosome in this problem is each potential route. The fitness score was calculated by adding up the distance between neighboring cities in the chromosome. The program will find the maximum fitness score in that generation to normalize the fitness score so the shorter path has a higher fitness score and the sum of all scores adds up to 1. Once the fitness score was calculated for

the current population, that population would perform a crossover depending on if a randomly generated number was bigger than the cross rate. The crossover method we used was randomly generating a starting index and ending index. We left the content between these two indexes and took out the part that's before the starting index and after the ending index, then filled empty slots with city orders from the mother chromosome. After the crossover, the program will generate another random number compared to the preset mutation rate to decide if these two children's chromosomes would be swapped partially for mutation. In the GA we implemented, we also added a selectivity variable for parents selection, we added this variable because we wanted to avoid children with really bad fitness scores. We found this selectivity variable stabilized the overall fitness score tendency during the reproduction process.
The Simulated Annealing Algorithm is based on annealing in metallurgy. Through controlled decreasing temperature, deciding if an algorithm should make a non-beneficial selection at any given time, we can find adequate solutions to difficult problems in a reasonable amount of time.
Ant Colony Optimization Algorithm is an algorithmic technique based on the path-finding behavior of ants searching for food. In nature, as ants travel from their colony to food, they search for the shortest path. If a path traveled is short the pheromone level on that path will increase, along with the probability that future ants will choose this path as well. When applied to the traveling salesman problem, the ants will spawn in different cities and travel from city to city using both distance and pheromones from other ants for direction. When the ants have completed their journey by ending in the city that they started in, we will use their path data (list of cities they travelled to) and their total distances to find the shortest path.

Experiments and Analysis
Genetic algorithm (GA):

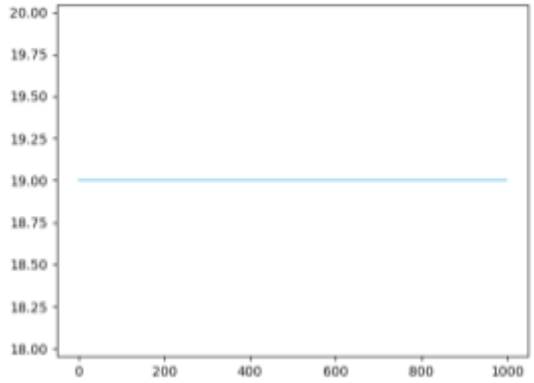


Figure 1: GA for five cities

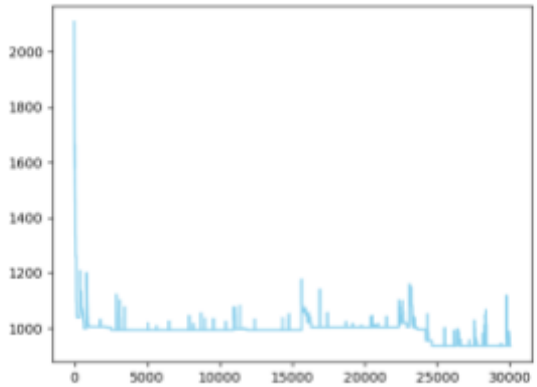


Figure 2:GA for twenty six cities

GA for 5 cities (Best score: 19):
The graph above is the result of GA running in 5 cities. It immediately found one of the best routes. For this case, we had 50 initial populations and 1000 iterations with a mutation rate of 0.3. GA was able to find solutions pretty easily due to the size of the sample cities. The graph was straight because we only select a percentage of the

population to reproduce the next generation. This condition helps stabilize the fitness score in bigger sample cities.

GA for 26 cities (Best score: 937):
In Figure 2, The best route found was: [3, 5, 4, 6, 7, 8, 9, 13, 14, 11, 12, 10, 15, 18, 19, 17, 16, 20, 21, 25, 22, 23, 24, 0, 1, 2] with the length of 937. This route has the best score for this dataset. For this, the population we set was 50. As we can see in the graph, it took around 25000 iterations to find the best route. We can also observe that the program was stuck in convergence many times. We assume it's because of the strict parents selection process, and the mutation rate is too low. This can be optimized by altering the selectivity and mutation variables.

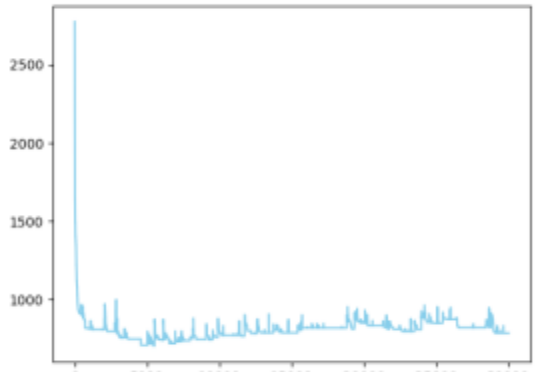


Figure 3: GA for forty two cities

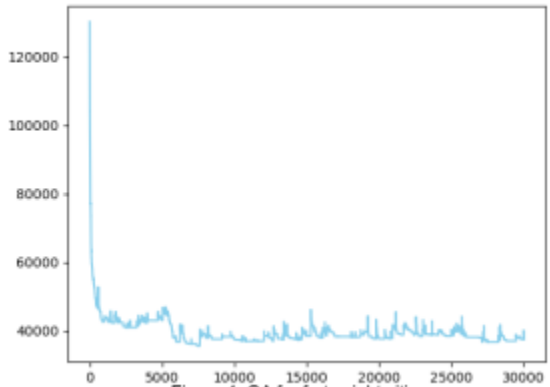


Figure 4: GA for forty eight cities

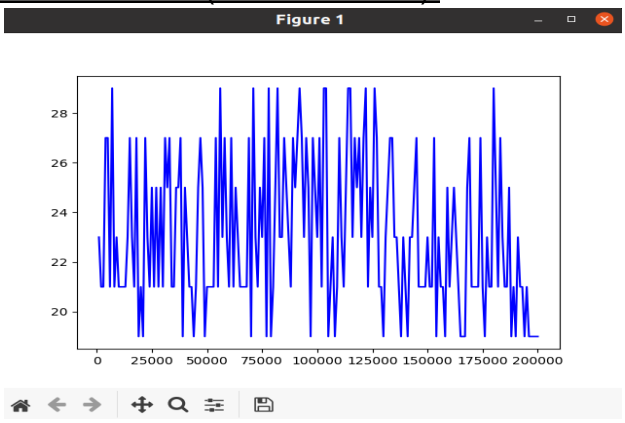
GA for 42 cities (Best score: 699):
In Figure 3, the best route found was: [31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 24, 25, 26, 23, 10, 11, 22, 21, 16, 12, 13, 14, 15, 17, 18, 19, 20, 27, 28, 29, 30] with a length of 704. It's not the best result, but it's the closest we could get in the 30000 iterations limit. From this point, algorithms like GA's weakness start to show. GA was really easy to narrow down the potential solutions, but it's hard to find the exact answers in more extensive scale conditions like this. We had to tweak around variables to really lower the average fitness score for each generation, which ended up setting the initial population to 70 with 30000 iterations. For mutation, we finally set it to 0.3. Higher than 0.3 caused really unstable generation scores; lower than 0.3 made the program stuck in really long convergence. We only select 15% of parents to reproduce because higher than this number caused fitness scores not to come down. I think if we keep running with more iterations, it will find the best route with these settings.

GA for 48 cities (Best score: 33523):
The best route found was: [18, 36, 5, 27, 6, 17, 43, 30, 37, 7, 0, 8, 39, 2, 21, 15, 40, 33, 13, 22, 11, 14, 32, 19, 46, 20, 38, 31, 23, 9, 44, 34, 3, 25, 41, 1, 28, 4, 47, 24, 12, 10, 45, 35, 29, 42, 16, 26] with a length of 35530. In this case we tuned down the mutation rate since we didn't see any sign of convergence according to Figure 4. All other variables are the same as 42 cities.

Analysis for GA:
We could see that GA is an excellent algorithm to narrow down the answers from those four experiments we did. It's also really efficient to find solutions for medium-size problems. However, we observed that it struggled to find the exact answer when the problem's size scaled up. From the graphs above, we could see GA was able to immediately lock down solutions within 10% of the best solution. But when the problem scaled up, it became difficult for GA to find an actual best answer. I believe if we just keep running it, it will find an optimal solution. But there's no guarantee how much time exactly would that take.

Simulated Annealing (SA):

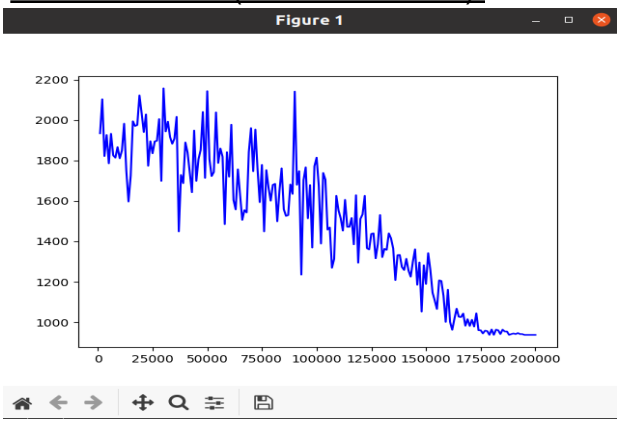
SA for 5 cities (Best score: 19):



This is the output for SA with five cities.
BEST TOUR: 19.0 FOUND IN ITERATION: 35
PATH: [3, 4, 1, 2, 0]

SA ran in 6.343 seconds

SA for 26 cities (Best score: 937):

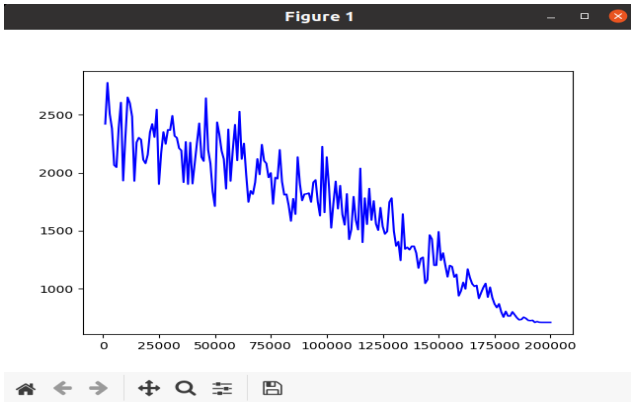


This is the output for SA with twenty-six cities.
BEST TOUR: 937.0 FOUND IN ITERATION: 152582
PATH: [21, 20, 16, 17, 19, 18, 15, 10, 12, 11, 14, 13, 9, 8, 7, 6, 4, 5, 3, 2, 1, 0, 24, 23, 22, 25]

SA ran in 27.480 seconds

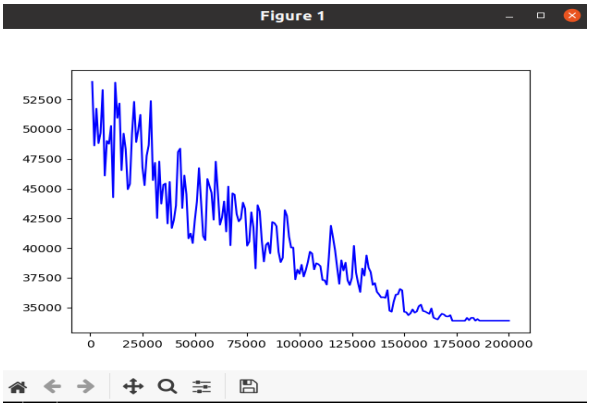
For five cities SA found the best path [3, 4, 1, 2, 0] with the shortest distance of 19 which matches the best possible score. The shortest path was found in iteration 35 and it took SA 6.343 seconds to complete this task. The temp mod was set to 3000 to achieve this result.
For twenty-six cities SA found the best path [21, 20, 16, 17, 19, 18, 15, 10, 12, 11, 14, 13, 9, 8, 7, 6, 4, 5, 3, 2, 1, 0, 24, 23, 22, 25] with the shortest distance of 937 which matches the best possible score. The shortest path was found in iteration 152582 and it took SA 27.480 seconds to complete this task. The temp mod (temperature modifier used for finding the temperature, temperature = [total iterations - iterations] / temp mod) was set to 3000 to achieve this result.

SA for 42 cities (Best score: 699):
For forty-two cities SA found the best path [28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 24, 25, 26, 23, 10, 11, 12, 13, 14, 15, 17, 18, 19, 16, 22, 21, 20, 27] with the shortest distance of 709 which is slightly over the best possible score of 699. The shortest path was found in iteration 192980 and it took SA 44.762 seconds to complete this task. The temp mod was set to 3000 to achieve this result.



This is the output for SA with forty-two cities.
BEST TOUR: 709.0 FOUND IN ITERATION: 192980
PATH: [28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 24, 25, 26, 23, 10, 11, 12, 13, 14, 15, 17, 18, 19, 16, 22, 21, 20, 27]

SA ran in 44.762 seconds



This is the output for SA with forty-eight cities.
BEST TOUR: 33888.0 FOUND IN ITERATION: 164079
PATH: [1, 28, 4, 47, 24, 13, 33, 40, 15, 21, 2, 22, 10, 11, 14, 39, 8, 0, 7, 37, 30, 43, 17, 6, 27, 5, 36, 18, 26, 16, 42, 29, 35, 45, 32, 19, 46, 12, 20, 38, 31, 23, 44, 34, 3, 25, 9, 41]

SA ran in 75.758 seconds

SA for 48 cities (Best score: 33523):

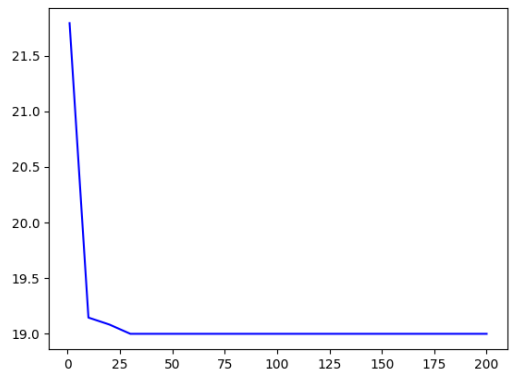
For forty-eight cities SA found the best path [1, 28, 4, 47, 24, 13, 33, 40, 15, 21, 2, 22, 10, 11, 14, 39, 8, 0, 7, 37, 30, 43, 17, 6, 27, 5, 36, 18, 26, 16, 42, 29, 35, 45, 32, 19, 46, 12, 20, 38, 31, 23, 44, 34, 3, 25, 9, 41] with the shortest distance of 33888 which is slightly more than the best possible score of 33523. The shortest path was found in iteration 164079 and it took SA 75.758 seconds to complete this task. The temp mod was set to 400 to achieve this result.

Analysis for SA:

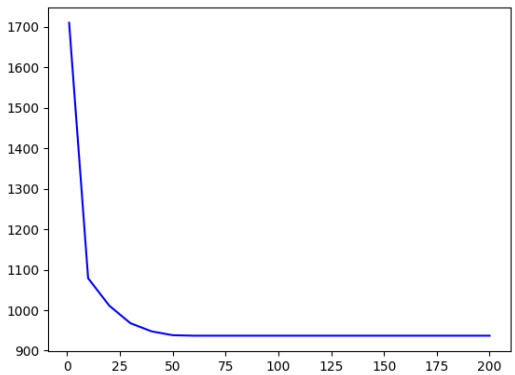
Simulated Annealing is no slouch when it comes to the Travelling Salesman Problem, handling 5 and 26 cities with ease and even though it did not match the best possible scores for 42 and 48 cities, it was only off by 1.43% (decimal 0.01430615164) and 1.08% (decimal 0.01088804701).

ACO results screenshots

ACO for 5 cities (Best score: 19):



ACO for 26 cities (Best score: 937):



FINAL GEN: 0200 AVG FIT: 19.00
Best Tour: 19.0 Found in Gen: 1
Path: [0, 2, 1, 4, 3]

GA ran in 0.707 seconds

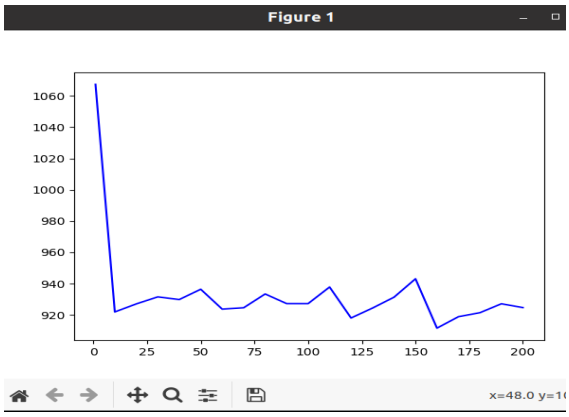
FINAL GEN: 0200 AVG FIT: 937.00
Best Tour: 937.0 Found in Gen: 3
Path: [0, 1, 2, 3, 5, 4, 6, 7, 8, 9, 13, 14, 11, 12, 10, 15, 18, 19, 17, 16, 20, 21, 25, 22, 23, 24]

GA ran in 12.124 seconds

The results for the ACO algorithm ran on 5 cities, showed the best route found was: [0, 2, 1, 4, 3]. In this case we have a colony size of 96, 200 iterations. Both the average path distance and the best path were found to be 19.

The results for the ACO algorithm ran on 26 cities, the best route found was: [0, 1, 2, 3, 5, 4, 6, 7, 8, 9, 13, 14, 11, 12, 10, 15, 18, 19, 17, 16, 20, 21, 25, 22, 23, 24]. In this case we have a colony size of 96, 200 iterations. The average path distance and the best path distance were 937.

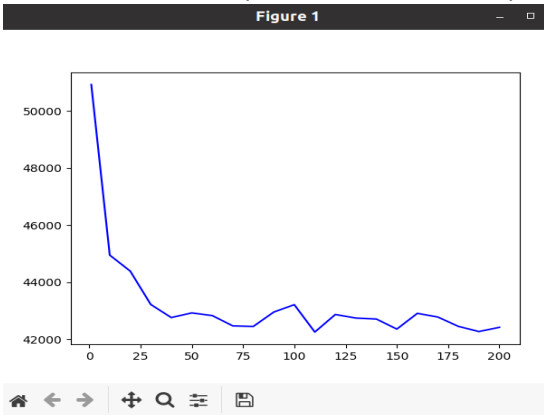
ACO for 42 cities (Best score: 699):



FINAL GEN: 0200 AVG FIT: 924.69
Best Tour: 704.0 Found in Gen: 11
Path: [35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 40, 0, 41, 1, 39, 38, 37, 36]

ACO ran in 164.394 seconds

ACO for 48 cities (Best score: 33523):



FINAL GEN: 0200 AVG FIT: 42423.60
Best Tour: 34728.0 Found in Gen: 157
Path: [28, 1, 25, 3, 34, 44, 23, 9, 41, 47, 4, 38, 31, 20, 46, 10, 11, 14, 39, 8, 0, 7, 37, 30, 43, 17, 6, 27, 5, 36, 18, 26, 42, 16, 29, 35, 45, 32, 19, 12, 24, 13, 22, 2, 21, 15, 40, 33]

ACO ran in 140.525 seconds

For 42 cities ACO found the best path [35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 40, 0, 41, 1, 39, 38, 37, 36] with the shortest distance of 704, slightly more than the best possible score of 699. The best path was found in generation 11 and it took ACO 164.394 seconds to accomplish this test.
For 48 cities ACO found the best path [28, 1, 25, 3, 34, 44, 23, 9, 41, 47, 4, 38, 31, 20, 46, 10, 11, 14, 39, 8, 0, 7, 37, 30, 43, 17, 6, 27, 5, 36, 18, 26, 42, 16, 29, 35, 45, 32, 19, 12, 24, 13, 22, 2, 21, 15, 40, 33] with the shortest distance of 34728, slightly more than the best possible score of 33523. The best path was found in generation 157 and it took ACO 140.525 seconds to accomplish this test.

Analysis for ACO:
Last but certainly not least we have Ant Colony Optimization, using the collective data gathering of our virtual ants to solve the Travelling Salesman Problem using a combination of direction and pheromone to seek out and report back the shortest path. ACO delivered a one-two punch for both the 5 and 26 city graphs, getting the best paths for both. ACO was close with 42 cities with a percentage of 0.71% (decimal 0.00715307582) but was not as accurate with 48 cities with a percentage of 3.59% (decimal 0.03594547027).

Challenges
When implementing the Ant Colony Optimization algorithm, we ran into a few problems and questions along the way. This algorithm made sense on paper but it was tricky to get this code to work because of the amount of variables to keep track of. There are a lot of concepts and pieces that go into this algorithm, we had to make sure all these pieces were used correctly to help determine the shortest path (and hopefully find that this path also contains the most pheromone). One question was how and when to increase the pheromone on a path, and when/how that pheromone dissipates. The initial pheromone weight and the evaporation weight are two variables we had to determine the value of, to find what amount would show the best results. We also had to go through the same process to find the number of iterations our algorithm would go through and the size of our ant colony. Spawning the ants was not an issue but then keeping track of their individual distances was a challenge because distances initially were wildly different from the actual shortest distance (getting the correct path but an incorrect distance value was irritating to say the least).
The GA initially did not perform well when we implemented the classic GA. We experienced that the fitness score would not steadily go down because the weights of cities were too close. When we implemented classic GA, we witnessed the average fitness score would not come down steadily. Fine-tuning the value of mutation did not help to stabilize the fitness score on the graph. Another problem we encountered was having too many variables for fine-tuning the reproduction process, plus it's hard to reproduce the same result twice when the sample map was huge. We tried to change one variable at a time, but each time, we had to run 10000 iterations multiple times to make sure we observe the right behavior after that variable was changed. It becomes really time-consuming to find the exact place to fine-tune and get a route as close as possible to the best route.

Future work
In this report we have explored the benefits and disadvantages of various AI methods for solving the traveling salesman problem. We have selected three of the most popular methods but further research can be done using the numerous other methods that are available. We think following things could be looked into further:

- Analyzing further on how variables in each method could affect the performance of their algorithm. We observed that tiny changes on even just one variable could shift the answer completely.

- More efficient implementation. All three implementations work as intended. But we could clean up our code more to reduce the running time such as reducing for loops for huge iterations.

Conclusion

In this project, we implemented three optimization algorithms to search solutions for the Traveling Salesman Problem. We used different sizes of maps with different variables to observe the behavior and the efficiency of each algorithm. Comparing each algorithm, we found that GA, SA and ACO all did incredibly well with 5 and 26 cities while all failing to get the best paths for 42 and 48 cities (GA and ACO ended up getting the closest to 699 with a score of 704 for 42 cities but SA was much closer to 33523 with a score of 33888 for 48 cities).