The Traveling Salesman Problem Approximation

GitHub: <https://github.com/chloeimbusch/DSAProject3>

Video:

The Problem:

The Traveling Salesman Problem (TSP) was first formulated in 1930. It is one of the most extensively studied combinatorial optimization problems in computer science and is a representation of many NP (nondeterministic polynomial time) problems. An NP is a set of problems with solutions that can be verified in polynomial time, but many take exponential time to solve.

This problem seeks to find the shortest Hamiltonian cycle that connects a series of points and is applicable to a variety of real-world optimization problems. Some of these examples include logistics routing, microchip design, and DNA sequencing.

The TSP and its algorithms are important because it is believed to be NP-complete.

This means that: 1. For any input the output is either yes or no.

2. If the answer is yes, it can be demonstrated through a polynomial length solution.

3. The correctness of each solution can be verified in polynomial time and brute-force

algorithm can find a solution by trying all possible solutions.

4. The problem can be used to simulate every other problem that can be verified in

polynomial time.

In relation to TSP, it is a combinatorial optimization problem, but currently the brute force method is currently the only algorithm that is guaranteed to find the optimal solution. This algorithm runs in O(n!) time, quickly becoming intractable for all but the smallest of datasets. Meaning that approximations of solutions must be used. If the requirement is relaxed from optimal to “good enough” many approximations can for found that run in polynomial, or super-polynomial, time which results in a solution that is on average within a constant of the optimal solution.

Features:

This project contains three main features.

1. The project is built on top of an interactive GUI that allows the user to select which approximation algorithm to run.
2. The implementation of the following algorithms: nearest neighbor, greedy heuristic, 2-Opt, genetic simulated annealing, ant colony optimization, and the brute force for comparison.
3. A visual representation of the effectiveness of the user’s chosen algorithm. This is done by displaying the best cycle found so far and graphing the distances of the best route that was found over iterations.

Dataset:

While at first it may appear that the dataset used is the geographic coordinates of the 200 highest populated cities in Florida, these are only the individual pieces that are used to compose dataset. The complete dataset is the sample space of all possible solutions. Routes are considered the same if they are the revers of each other, therefore the sample space contains possible solutions. N represents the 200 cities being traveled between, resulting in approximately different possible solutions. For perspective, the number of Planck times since the big bang is estimated to be around and estimates for the number of atoms in the known universe is approximately . If these two numbers were multiplied together, it is still only a small fraction of the problem’s sample space. This number is the primary issue that any traveling salesman problem approximation algorithm will need to overcome.

Tools:

This project requires Python 3.7 or newer and was written using 3.10.9. Two primary external libraries were utilized. Pygame was used to build the GUI, and matplotlib was used to plot how the calculated distance of the algorithm improves over time.

Algorithms:

* Nearest Neighbor
  + A greedy algorithm that begins by selecting a starting city, then iteratively finding and adding the city that is closest to it. This implementation runs the algorithm in parallel and by default has 100 different agents running the same algorithm with the goal of selecting different starting cities.
* Greedy Heuristic
  + This algorithm operates similarly to Kruskal’s MST algorithm. Every iteration it adds the shortest valid edge to the overall route. The main difference between the algorithm is that only endpoints of the constructed are considered to be valid vertices to connect to.
* 2-Opt
  + This algorithm is a variant of the K-opt algorithm. This set of algorithms works to improve an existing solution, rather than generate one. During every iteration this algorithm attempts to identify segments that can be swapped with each other to create a shorter route and makes the swap if the new route is an improvement over the old route.
* Genetic
  + Genetic, or evolutionary, algorithms attempt to solve a wide range of optimization problems by mirroring biological processes. A population of individuals have their fitness measures and then based on this are selected into a mating pool. They are then “mated”. For the TSP individuals are represented by the individual different possible solution, and “mating” by two route swapping sections of their route. Lastly, each individual undergoes a mutation process where cities are randomly swapped.
* Simulated Annealing
  + Based on the metallurgical annealing process, this algorithm stochastically changes the current route and compares the original route to the changed one. If the changed route is better, it is accepted as the new one. The annealing part of this algorithm occurs if the new route is worse. The new route is accepted based on the equation . The important pieces of this equation are the ∆E, which is the difference in the fitness of the new and old solutions, and t which is the temperature. This temperature variable is decreased as the algorithm is iterated, and as a result, the worse solution is selected at a higher rate early in the process, and the better solution is more likely to be retained the longer the algorithm iterates.
* Ant Colony Optimization
  + This a probabilistic algorithm that utilizes the concept of pheromone trails to approximate a solution. First, a number of “ants” are created starting from a random city. They will then select their next city based on both the strength of the pheromones and distance to the other cities. Once each ant has selected a route it will deposit additional pheromones on its chosen connections. This approach is similar to nearest neighbor but has an additional layer of randomness that allows the algorithm to better explore the solution space in search of the globally optimal solution.
* Brute Force
  + An algorithm that iterates through every possible solution until it finds the best one.

Additional Data Structures and Algorithms

Several of the implemented algorithms utilize additional data structures and algorithms under the hood. The nearest neighbor algorithm utilizes a series of linear searches to find the closest city. The strength of pheromones between cities in the ant colony optimization is represented by a weighted adjacency matrix. Additionally, memorization is utilized when calculating the distance between cities, which is a very common operation in this project.

Roles and Responsibilities:

We utilized [agile methodology](https://trello.com/b/c0Zj9Dev/dsa-project-3) to divide the workload. We had an initial meeting where the course of the project was planned out, various tasks were added to the linked tracker, after which we each claimed and worked on a task when we had time available. Over the course of the project, Chloe implemented the greedy heuristic algorithm, the 2-Opt algorithm, created the video, and provided the foundation of the workflow by maintaining the git repo and the agile storyboard. Kyle implemented the GUI in addition to the nearest neighbor, genetic, simulated annealing, ant colony optimization, and brute force algorithms.

Analysis

Changes:

This project mostly proceeded as was initially planned, though there were some changes. The 3-opt algorithm was dropped as it was too similar to the 2-opt algorithm. The simulated annealing and ant colony optimization were added for their interesting implementations.

Time Complexity:

* Nearest Neighbor
  + O(m\*n2), where n is the number of cities in the route, and m is the number of parallel nearest neighbor agents being run simultaneously.
* Greedy Heuristic
  + This algorithm runs in O(n2logn), where n is the number of cities in the route. In the worst case, this algorithm would iterate through every edge in the fully connected graph consisting of every city in the route. This algorithm also requires sorting these edges, which is where the logn comes into the picture.
* 2-Opt
  + The time complexity of this algorithm is dominated by the time complexity of running the initial greedy solution, which is O(n2logn). The actual 2-Opt portion of the algorithm runs in O(n2) time, as in the worst case, it would need to iterate through all edges in the route.
* Genetic
  + O(m\*n\*p), where m is the number of generations, n is the number of cities in the route, and p is the number of individuals in the population.
* Simulated Annealing
  + O(m\*n), where m is the number of iterations, and n is the number of cities in the route.
* Ant Colony Optimization
  + O(a\*m\*n2), where a is the number of simulated ants, m is the number of iterations the algorithm runs, and n is the number of cities in the route.
* Brute Force
  + O(n!) where n is the number of cities in the route, as discussed in the dataset section.

Reflection

Overall Experience:

Now that it’s done, we can take a step back and say that things went relatively smoothly, in the grand scheme of things anyways. We encountered many, many problems, but we were mostly able to overcome or work around them. Some of the algorithms were frustrating to implement, and others irritated us by stubbornly resisting improvements and other optimizations. But overall, this was a fun project that we hope to continue working on and adding more features to.

Challenges:

One challenge that we faced was that neither of us had ever written a GUI for a project of this size before and we had a lot of early struggles with getting the various required pieces to fit together smoothly. It eventually came together, and we were able to rewrite various components so that they were more interchangeable, which made further modifications and additions simpler.

An additional challenge was that several of the algorithms, namely the genetic, ant colony optimization, and an unincluded particle swarm optimization, experienced pre-mature convergence. Roughly speaking, all these algorithms work by initializing many random solutions in the sample space and moving the various solutions towards the current best solution. This allows these algorithms to be useable in a wide variety of situations, but in order to be effective, these algorithms require that enough of these solutions be created to effectively cover the sample space. Pre-mature convergence occurs when these solutions all gravitate towards a local minimum and become similar to each other, limiting their ability to break out of the local minimum. A general recommendation that we encountered was that the number of sample solutions (individuals in the genetic algorithm, ants in the ant colony, and particles in the particle swarm) should greatly exceed the number of cities in our route. This was somewhat possible to achieve for the computationally simpler genetic algorithm, but was not achievable in the ant colony, and was so bad in the particle swarm optimization that we chose not to include it. It appears that this is the price that these algorithms pay for their generality.

A final challenge was scope creep. While researching this problem, we encountered a significant number of interesting sounding algorithms that distracted us from focusing on the core project. In addition to the ones that we did eventually implement, we attempted to implement particle swarm optimization, and looked into state transition, tabu search, artificial bee colony, and black hole optimization.

Hindsight Changes:

Based on the challenges we encountered, we would spend more time upfront planning how the various pieces of the GUI will fit together, in order to reduce the coupling between the components. Additionally, prior to beginning to work on the various algorithms, we should have spent more time researching what requirements they have for working well. This could have saved us the time spent working on the particle swarm optimization algorithm and may have dissuaded us from attempting the ant colony optimization.

Lessons Learned:

* Chloe: I had personally never heard of the traveling salesman problems, so this was a really good learning experience. I got to see several algorithms I had not heard of in work and was able to do a lot of deep diving and research about the topic.
* Kyle: Learning how the various algorithms worked was a beneficial experience, both from simply experimenting with these solutions in general, and also comparing how the different algorithms approached the same problem. I’ve hopefully added a few algorithms tricks to the proverbial tool chest. The GUI implementation was also surprisingly helpful. There were a lot of pieces to fit together into a cohesive whole, and seeing what eventually worked to fit them all together is something I can take away for the next project of this size.

Sources

* Starting point for genetic algorithm: <https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35>
* Starting point for simulated annealing: <https://medium.com/@francis.allanah/travelling-salesman-problem-using-simulated-annealing-f547a71ab3c6>
* Source for memorization of distance calculations: <https://codereview.stackexchange.com/questions/110221/tsp-brute-force-optimization-in-python>
* Source code for greedy heuristic and 2-opt: <https://colab.research.google.com/github/norvig/pytudes/blob/main/ipynb/TSP.ipynb#scrollTo=IhWcbRbp6pE1>