**Traveling Salesperson Problem Optimization**

William Narciso, Joey Bursey, Samuel Vilchez, Mowhebat Bazargani

***Introduction:***

The traveling salesperson problem (TSP) is a classic optimization problem that tries to solve the issue when given a list of cities and distances between each city, find the shortest possible route that visits each city exactly once and returns to the origin city.

The traveling salesperson problem or TSP is classified as a problem that is NP-Complete as it takes exponential time to solve and the solution cannot be checked in polynomial time. The traveling salesperson problem is therefore an NP-hard problem in which the complexity of calculating the best route to all cities increases as we add more destinations to the problem. One could imagine a brute force method which would check every possible permutation to find the shortest path. However, this brute force method would take O(n!) time. Even in a small case of 48 states, iterations are required. Since changing the order of two cities changes the path length seemingly randomly, each iteration must be checked to guarantee a shortest path. So, we need to find a better method.

There are many similar problems to the TSP that would benefit from solutions to the TSP such as amazon delivery routes going to various destinations. Solving the TSP would provide solutions that can be applied to similar problems or modified to suit their needs.

In this project we will be examining several different known optimization methods, testing their performance and comparing the results. We will use the following five optimization methods in order to solve the Traveling Salesperson Problem: Nearest Neighbor, Step-by-Step, Simulated Annealing, Genetic Algorithm and Particle Swarm Optimization methods.

***Credit for Coding:***

Joey: Nearest Neighbor and Step-by-Step

Mowhebat: Simulated Annealing

William: Genetic Algorithm

Samuel: Particle Swarm

***Coding and Testing:***

We used the coordinates of the 48 mainland capitals of the US (att48.tsp). We define the cities using x and y coordinates, and define the distance between the pair of cities to be the Euclidean distance. The shortest route possible for the 48 capitals of the US is 10,628.

We chose to code the nearest neighbor and step by step algorithms in C++ since both are relatively easy to implement and for the ability to optimize code in a lower level language. Nearest neighbor is the simplest algorithm we looked at. It takes a starting city, in this case city 0, and finds its nearest neighbor. This process repeats until every city has been visited. When coded, the algorithm looks like Selection Sort, where the selected element is the nearest neighbor to the previous city. So the algorithm runs in O(n2) time. For a few 48 elements, the algorithm runs in less than one second. However, it is quite terrible at finding a fast route. Considering the point of view of the salesperson, it might make sense to travel to the next closest city. After all, he gets to sell his wares faster. This algorithm, though, returns a path length that is often 681% larger than the shortest path length. For the set of 48 states, we got a path length of 83,039.59. It is possible that Nearest Neighbor will perform better on a smaller set of points, but larger sets allow for more pitfalls.

The step by step algorithm fared a little better. This algorithm starts with a random permutation of the cities, which was achieved by shuffling the array of cities. On each iteration of a loop, it chooses two random cities and checks if swapping them provides a shorter path length overall. If it does, swap them, if not, keep the current permutation. We chose to loop 2000 times, a number sufficiently larger than 48. Due to the random nature of the algorithm, we will likely get a different path each time, and we have the added challenge of getting caught in local minima. To get around these issues, we chose to run the algorithm 100 times to get a reasonable data size. The data converged quite nicely to a path length that had a percent error of 417%. It had a mean of 55,048.79 with a standard deviation of 4553.27. The only downside when compared to the nearest neighbor algorithm is the run time. Since the algorithm has to check the path length each iteration, it becomes slow with a large number of iterations. In this case, 200,000 total iterations took around 7 seconds. It is notable that this algorithm should scale nicely. The time sink is in the number of iterations, not necessarily the number of points. Adding more iterations did not show an improvement in the path length, so it is safe to say that the algorithm is optimal for what it can do.

For the genetic algorithm and simulated annealing method, we used mlrose (Machine Learning, Randomised Optimization and Search) package. The *mlrose* package is a Python package for applying some of the most common randomized optimization and search algorithms to a range of different optimization problems, over both discrete- and continuous-valued parameter spaces.

In conventional methods, we update a point when a function has a lower value than the current point. This approach leads us to a local minima. Simulated annealing avoids local minima by accepting higher value for updates with some probability. This probability decreases over time, which means we accept wrong values at first with higher probability, and this probability is going to decrease by a small amount after each iteration. Therefore, we can seek the global minima with simulated annealing. This is a key advantage over the Step-by-Step algorithm which we will demonstrate later. We tuned the simulated annealing method with a max\_attempts of 500 (max\_attempts is the maximum number of attempts to find a better neighbor at each step). Increasing the maximum number of attempts up to 1000 did not result in a shorter path. This method gave us a shortest route of 48,459.59, which is a relatively good answer in 14.05 seconds.

For the genetic algorithm we create an initial random population which at each step uses members of the current generation to create the next population. The new population is created as it evaluates a fitness value for each member of the current population. The best member is chosen and used to create the next population. Random changes are made to the children via mutation or by combining traits from a pair of parents. This is called a crossover event. We tuned the parameters of the genetic algorithm with a population size of 150, mutation probability of 0.2 and a max\_attempts of 10. Increasing the maximum number of attempts up to 100 did not result in a shorter path. This method gave us a shortest route of 113,973.69, which is considerably larger than the shortest path possible. Increasing the max\_attempts for the genetic algorithm does not have much noticeable effect on the shortest route.

For the particle swarm method we used the particle and PSO classes defined in tsp\_pso to run the traveling salesperson problem. In the particle swarm method we have a population that is composed of particles. Each particle has its position, velocity, and its personal best location it has found. These particles will get updated making their way towards a global best position found. Our particle can be viewed as a path in the traveling salesperson problem. Representing velocity in terms of paths was a bit hard to grasp in terms of code. The article “Solving City Routing Issue with Particle Swarm Optimization” made understanding the concept of velocity in terms of the traveling salesperson problem simpler. The article proposed using a sequence of swaps on a particle and showed a modified velocity update equation where all swap operators should be maintained according to probabilities α and β. The particle swarm method in tsp\_pso uses this modified velocity update equation to update an individual particle.

When testing values for the population size and the number of iterations, it was evident that increasing these two values would not have much effect on improving the optimal solution. When a value of 1000 iterations and a population of size 20 was used, the solution was found relatively quickly. Increasing these two values did not have much effect on improving the solution but rather increased the running time of the algorithm.

***Results:***

We can compare the shortest route and the time elapsed to compute each optimization method. Each of these results are the best possible for each algorithm. Adding more time or iterations does not make a noticable difference.

|  |  |  |
| --- | --- | --- |
| **Method** | **Time Elapsed (s)** | **Minimum Cost Route** |
| Nearest Neighbor | < 1 | 83,039.59 |
| Step-by-Step | ~7 | 55,048.79 4553.27 |
| Genetic Algorithm | 11.44 | 113,973.69 |
| Simulated Annealing | 14.05 | 48,459.59 |
| Particle Swarm | 2.72 | 101,595 |

***Conclusions:***

Overall, Simulated Annealing proved to be the best algorithm for finding the shortest path. We found that with a decent run time of 14.05 seconds, it could find a path length with 355% error. Step-by-Step took a shorter amount of time, but found a slightly worse path with 417% error. Going by path length, Nearest Neighbor performed the third best. Because of its simplicity, Nearest Neighbor makes for a quick and dirty solution for real world routing problems. Despite this, it should not be used. The algorithm is highly dependent on the locations of the cities. It could be possible to create a set of cities specifically to make the Nearest Neighbor algorithm run poorly. Therefore, it should not be considered a stable algorithm. Particle Swarm demonstrated a tradeoff between time and path length. It ran almost as fast as Nearest Neighbor, but found an even worse optimal path with 681% error. Lastly, the Genetic algorithm proved to be the worst of the five algorithms we tested.

The Genetic Algorithm and Particle Swarm both performed poorly when optimizing the Travelling Salesperson Problem. This is likely due to the random nature of how changes in the path order change the total path length. Of course, it isn’t really random, just hard to predict. Both of these algorithms try to find a shortest route based on the knowledge of previous routes that were also short. These attempts were foiled because even though the paths were similar between iterations, the path length could change drastically. This is one of the main issues with solving the Travelling Salesperson Problem. Solutions must be found in seemingly random ways.

It is reasonable then, that our two best algorithms were Simulated annealing and Step-by-Step. They both chose random steps to try and make the path shorter. Step-by-Step did this in a very elementary way, only allowing itself to shorten the path, and only trying once per iteration. Simulated Annealing, by comparison, allowed itself to take worse steps with a dynamic amount of probability that decreased with the number of iterations. Since Simulated Annealing performed better in the end, it further confirms our suspicions that there would be local minima to trap the algorithms. Simulated Annealing had a better way of avoiding the local minima, so it performed better.

None of these algorithms managed to find the true shortest path. They offer solutions that are decently close to optimal, but they still have room for improvement. Furthermore, letting the algorithms run longer does not provide a shorter path. This is the best they can do. The optimal result remains to be solved by brute force.

Of course, a no soliciting sign would solve your travelling salesperson problem too.

References

Alhanjouri, Mohammed. (2017). “Optimization Techniques for Solving Travelling Salesman

Problem.” *International Journal of Advanced Research in Computer Science and*

*Software Engineering*. 7. 165-174. 10.23956/ijarcsse/V7I3/01305.

Castro de Souza, Marcos. “marcoscastro/tsp\_pso.” *GitHub*, 30 Apr 2015,

github.com/marcoscastro/tsp\_pso/blob/master/tsp\_pso.py

Gao, Yuan. “Heuristic Algorithms for the Traveling Salesman Problem.” *Medium*, 10 Feb. 2020,

medium.com/opex-analytics/heuristic-algorithms-for-the-traveling-salesman-problem-6

53d8143584.

Hayes, Genevieve. “Gkhayes/Mlrose.” *GitHub*, 2 Nov. 2019, github.com/gkhayes/mlrose.

K. Hadia, Sarman, et al. “Solving City Routing Issue with Particle Swarm Optimization.”

*International Journal of Computer Applications*, no. 15, Foundation of Computer

Science, June 2012, pp. 30–38. Crossref, doi:10.5120/7266-0348.

Kochenderfer, Mykel J., and Tim Allan. Wheeler. “Algorithms for Optimization.” *The MIT*

*Press*, 2019.

S. S. Juneja, P. Saraswat, K. Singh, J. Sharma, R. Majumdar and S. Chowdhary, "Travelling

Salesman Problem Optimization Using Genetic Algorithm," 2019 Amity International

Conference on Artificial Intelligence (AICAI), Dubai, United Arab Emirates, 2019, pp.

264-268, doi: 10.1109/AICAI.2019.8701246.

*TSPLIB*, 19 Feb. 1997, elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html