PES University

Comparison and Analysis of TSP Algorithms

Advanced Algorithms

Taarun Sridhar
11-18-2022

Abstract

One of the most well-known optimization issues is the "Travel Salesman Problem." While an optimal solution cannot be found, non-optimal alternatives get close to being ideal and keep execution times short. The most popular algorithms for solving this problem are compared in this research in terms of route length, elapsed time, and iterations. Different scenarios are used as examples while simulating the TSP, and each case's convergence is examined.

Introduction

Since Karl Menger first proposed the Travel Salesman Problem (TSP) in 1930, it has grown to be one of the most extensively researched issues in optimization. The issue is described as follows: given a set of cities, the travel distance (cost) between each pair of cities, and a single salesperson, the ideal route should be calculated for the salesperson to follow in order to visit each city precisely once, make a return trip to the starting city, and do so at the lowest possible cost. The uses for TSP go far beyond straightforward salesmanship and include vehicle routing, logistics, printing, and PCB board soldering. TSP is regarded as non-deterministic polynomial-time challenging (NP-hard). There are both precise and approximative solutions to solve it. According to the TSP precise solutions, every permutation combination must be tried with an O(n!) complexity. Because of this, the running time combinations are 3628800, which is impractical even with a modest number of cities (n=10). The approximate solution, on the other hand, employs heuristic algorithms to reduce complexity and discover an approximation to the ideal answer in fewer steps. Even though the heuristic solution (shortest path and smallest cost) is not perfect, it is nevertheless deemed valuable because it takes less time than the perfect solution. The solution may also be optimized by combining the heuristic algorithms with additional optimization algorithms.

Literary Review

Heuristic Algorithms to Solve TSP

The TSP is solved using a variety of algorithms in dynamic programming. Due to the potential impossibility of finding the ideal answer (huge number of cities), Held-Karp lower bound is used to assess an algorithm's performance. In other words, the Held-Karp boundary describes the distance between a given solution and the optimal solution for a certain method. The relaxed answer to the linear programming of the TSP is the Held-Karp bound. Typically, the HK lower bound is 0.8% shorter than the ideal tour length.

Nearest Neighbor

It is the most straightforward heuristic algorithm for solving TSP. This is a brief summary of the algorithm:

1) Choose a random city n and set it as the starting city n0.

2) Locate the closest unexplored city and visit it.

3) Enter the present location as a visit

4) Exist any unexplored cities? Yes, proceed to (2)

5) Go back to the beginning location. The map of the United States is taken into consideration when assessing the closest neighbor algorithm solution. 9.9 million square kilometers make up its area. The rectangle's dimensions are 2545 km in length and 4313 km in breadth. Over that area, the cities are strewn around at random. Within a 4313 by 2545 km rectangle, 20 cities (nodes) are allocated at random. Calculating the best path to visit each city only once before heading back to the starting point is the objective. The fact that the cost of travel from city A to city B is precisely the same as that from B to A makes this example a symmetric TSP, it is important to note. The asymmetric TSP takes into account the unequal travel expenses from both places; however, it is outside the focus of this article. The ideal solution, which has a combined total length of 4616 km, is depicted in the diagram below. Even though this approach required the most iterations and complexity, it discovered the shortest path. Despite having a greater total length (15800 km), the solution can be found in O (N^2 $\log_2(N)$) iterations, where N is the total number of cities that need to be visited. Within 25% of the Held-Karp lower bound, the solution is kept by its closest neighbor.

2-Opt Neighbor Search

After generating a tour with the nearest neighbor algorithm, a series of exchanges is performed to improve upon the original tour. To use the 2-opt heuristic, the first step is to generate an initial tour. The original approach was to run the nearest neighbor algorithm repeatedly to find the starting point that resulted in the best initial tour, but it is found that the best starting tour didn't always result in the best improved tour after running the 2-opt. Instead, for relatively small test cases it is resorted to running the entire optimization, including the 2-opt, with each possible start point and choosing the best one. The basis of the 2-opt algorithm is to swap the endpoints of two edges of a tour and see if the resulting tour has a lower distance. For example, in a tour with cities A-BC-D~, where ~ represents an arbitrary number of intermediate cities, a 2-opt exchange between edge AB and edge CD would result in the tour A-CB-D~, with the cities between C and B being in reverse order compared to the original tour. In its naïve form, the algorithm iterates through each city on the tour, performing a 2-opt exchange between each of the other edges and retains the change only if it reduced the total tour distance. This unoptimized version of the algorithm runs through every 2-opt exchange, resulting in fairly poor runtime. To improve on the runtime of the naïve 2-opt algorithm, there is a comparison that exploits the triangle inequality to determine if a swap will reduce distance [3]. This is the method used to improve the speed of the algorithm. Using the same notation as above, the comparison is that dist (A, C) must be less than dist (B, D) for an exchange to be distance-reducing. Using this comparison, the number of exchanges needed to achieve the same result is reduced.

Genetic Algorithm

This algorithm is renowned for resolving challenging issues without an obvious ideal answer. It is comparable to how natural selection works. Every person in the population has their fitness function calculated by the algorithm. The population is then expanded by producing new individuals. Like the natural genome, it uses mutation to introduce randomness into the process. The person (solution) with the highest fitness function is finally chosen. Certain restrictions are necessary when using genetic algorithms with TSP. A shorter path can be found via the Genetic Algorithm with more iterations than the closest neighbor (11900 km).

Greedy Heuristic Algorithm

This algorithm falls under the domain of heuristic algorithms, which look for local optima and improve the local optimum solution in order to find the global optima. It starts by sorting every edge before choosing the one with the lowest cost. As long as no loops emerge, it keeps choosing the best options for the following step. The greedy algorithm has an O (N^2 $\log_2(N)$) computational complexity, and there is no assurance that a global optimum solution will be discovered. The greedy method, on the other hand, keeps the answer within 15% to 20% of the Held-Karp lower constraint and ends after an acceptable number of steps. Despite having a somewhat smaller overall distance than the nearest neighbor technique, this solution still requires more complexity and time to execute. It is important to note that all algorithms converged as a result of the few cities and thus few route probabilities. As a result, they all lead to a viable (i.e., not too long) solution.

TABLE I

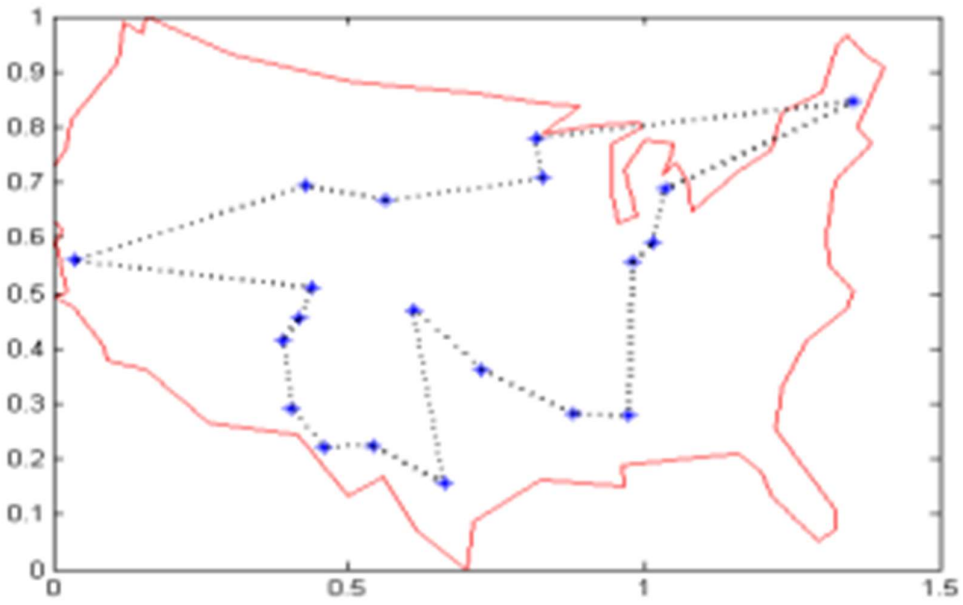
TSP OF 100 CITIES, ALGORITHM COMPARISON

Algorithm	Optimal route length (km)	Elapsed time (sec)	Iterations
Nearest Neighbor	26664	2.5	100
Genetic	25479	45	10000
Greedy Heuristic	23311	0.07	18

TABLE II

TSP OF 1000 CITIES, ALGORITHM COMPARISON

Algorithm	Optimal route length (km)	Elapsed time (sec)	Iterations
Nearest Neighbor	83938	95.5	1000
Genetic	282866	468	10000
Greedy Heuristic	72801	127	151



TSP Examples of 20 cities: Optimum Solution

Note: Data and images taken from references as quoted below

Conclusion

The Travel Salesman Problem is discussed in this essay. Conclusion: Even though the Greedy Heuristic requires more iterations to solve the TSP, its outcome is the most closely related to the ideal answer. With a reasonable number of iterations, the result is still attainable. The Genetic Algorithm, on the other hand, provides a longer-distance option despite failing to locate the shortest path. This is not a surprising outcome given that the genetic algorithm uses permutations between cities to determine the best path, but since these permutations are random, there is no assurance that the path chosen is the best one. Therefore, it can be said that the Genetic algorithm does not converge to a good solution when there are a lot of nodes.

GitHub Link

https://github.com/taarunsridhar/advanced-algorithm-project

References

- Comparison of TSP Algorithms: Project for Models in Facilities Planning and Materials Handling, Dec 1998

- Angeniol, B., Vaubois, G de L C. and Texier JYL.(1988) Self-Organizing Feature Maps and the Traveling Salesman Problem, Neural Networks, Vol 1, pp 289-293
- Heragu, Sunderesh (1997) Facilities Design, PWS Publishing Company, Boston, M
- M. M. Flood, “The Traveling Salesman Problem,” Opns. Res., 1956
- C. Nilsson., Tewfik, “Heuristics for the Traveling Salesman Problem,” Linkoping University, pp. 473-480, June 1996
- D. Johnson, L. McGeoch, The Traveling Salesman Problem: A Case Study in Local Optimization
- https://www.ijeat.org/wp-content/uploads/papers/v4i6/F4173084615.pdf