A Genetic Algorithm for Solving The Traveling Salesman Problem

Intro

In the scope of problems in computer science, the Traveling Salesman Problem (TSP) is one of the most famous and deeply explored problems. The basic idea is best formulated as a weighted graph where cities are represented on a map. The problem for a salesman is how can he get to each of these cities (nodes) in the most efficient (cheapest) way possible, without ever visiting a city more than once? At first glance it seems simple to solve this through brute force, simply calculate the cost of every permutation of routes through the graph and select the cheapest one. Unfortunately for most systems this becomes obstructively difficult, bordering impossible to do once you have more than twenty cities. More creative solutions have been developed over time, but there is yet to be discovered a polynomial time algorithm for finding the optimal solution. Most algorithms are centered around finding a good solution, but unless you have the cost of every permutation it is not possible to know if this is the perfect solution to your salesman's problem.

Recent History

What makes this problem interesting is the combination of being NP-Hard (no polynomial time solutions) and the diverse plethora of incredibly clever algorithms that have been devised to solve it, algorithms that can be applied to other optimization problems. There exist some exact algorithms that can find the most optimal solution for hundreds of cities, some up to even thousands. The largest instance of a solved traveling salesman problem is

for every city in the entire country of Sweden (24,978 cities) by proving no shorter tour exists, this was in 2004. The record was smashed in 2006 for a VLSI application with 85,900 locations using the Concorde codebase of 700 functions that were customized for this specific TSP, and has been used to solve other TSPs.

Ant Colony/Swarm

Ant colony optimization (ACO) is a heuristic algorithm which has been proven a successful technique and applied to a number of combinatorial optimization problems and is taken as one of the high performance computing methods for Traveling salesman problem (Hlaing). Ant colony optimization has downsides in that it can cost too much time to converge and gets trapped in local optima, but remains a very good search for optimization problems (Hlaing). An improved version of the ant colony optimization in Hlaing's paper improves both the convergence speed and improves the performance by using a dynamic updating rule for heuristic parameter based on entropy (Hlaing). Using swarm intelligence theory, researchers developed an optimization algorithm called chaotic ant swarm to find global optimums in a search space, although it could not solve TSP directly. Other researchers used this to base a variant, chaotic ant swarm-traveling salesman problem. This generates the optimal solution for TSP problems all the way up to sizes of 150, making it a useful tool and a very competitive heuristic (Zhen). The algorithm is inspired by the actual behavior of ants in nature and chaos theory, each ant in the search hunts and interacts with neighbors using pheromone trails to locate the global optimum (Zhen). The original methodology was implemented by Marco Dorigo and Luca Maria Gambardella, in their paper they describe in detail the way nature was used to simulate artificial life to formulate the algorithms used in future ant colony research projects (Dorigo).

Genetic Algorithms

This is the method I have chosen to analyze for solving the Traveling Salesman Problem. A genetic algorithm is a heuristic based on the natural process of evolution. They depend on two main variation operators, crossover and mutation (Contreras-Bolton). A modified genetic algorithm for solving the traveling salesman problem utilizing multi-operators showed that synergy between operators was mutually complementary (Contreras-Bolton). The

process for searching for good combinations and ultimately a good solution candidate was shown to be more effective and better than using it than a single operator or the operators mentioned earlier (Contreras-Bolton).

Bird Colony/Particle Swarm

Particle swarm optimizations are a type of algorithm that tries to improve a candidate solution with regards to a heuristic where a pool of candidates (particles) are moving around in the search-space, under mathematical position/velocity formulas (Yan). Each particle moves towards its own best known position but is also guided by the surrounding particles, and the swarm as a whole moves towards the optimal solution. Unlike genetic algorithms, particle swarm doesn't rely on selection operators, crossovers, mutations, and other genetic operators. The population is optimized directly through information exchanged between individuals (Yan).

As you can see from the previous sections, solutions to the Traveling Salesman Problem are heavily influenced by nature. From the ant colony, bird colony, and evolutionary algorithms, all pose solutions with different benefits and drawbacks. Selecting a methodology is dependent on the size of your data and the resources you have allocated to solving it; and even within the individual methodologies are hybrid and modified algorithms, as the Traveling Salesman Problem continues to be researched and explored.

Genetic Algorithm Implementation

This section will go more in depth into the actual genetic algorithm that will be analyzed for solving the traveling salesman problem. Although some powerful algorithms like the swarm are quite good at finding a near-optimal solution, the genetic algorithm will be easier to understand and analyze here. For solving the TSP via genetic algorithm, a few special considerations need to be made. The first caveat being that a solution can only represent a path that goes through every location a single time, any more or less and it unqualified. This means that the mutation and crossover methods in the genetic algorithm must be modified. First of all the mutation should not be able to add or remove locations, creating an unqualified route. It should only be able to shuffle around the order of the locations toured without modifying the places involved. This means a swap mutation would be used, simply swapping cities around with some random variation; and this will never create an invalid route. The second consideration is using a method

of crossover that won't produce invalid solutions. The method used in this genetic algorithm is selecting a sublist of the first parent and adding it to the offspring, then any missing values are added to the offspring from the second parent so that the number of locations is preserved.

Experimental Design/Results

The genetic algorithm could be manipulated in a variety of ways to determine its effectiveness and observe exactly how it works. For testing purposes twenty cities will be used and represented via two coordinates so that it is easy to calculate their relative distances from each other. After the initial populations (list of routes) are randomly produced, the cheapest one is printed out as the Initial distance. For table 1 you can see the algorithm run on twenty cities with the following default settings: Population of 50, 100 evolutions, and a mutation rate of 0.015. The variation between different runs is accounted for by the randomness inherent within genetic algorithms, since the cities and their relative distances were all kept constant. Three separate modifications were made: In table 2 the evolutions were increased to 200, in table 3 the population was dropped to only 5 individuals, and in table 4 the mutation rate was increased to 0.05.

Table 1: Default Settings

Trial	Initial Distance	Final Distance
1	1616	1155
2	1930	1228
3	1873	994
4	2009	1177
5	1780	998
6	1905	958
7	1940	925
8	1782	984
9	1905	935
10	2031	1020
AVG	1877.1	1037.4

Table 2: 200 evolutions

Trial	Initial Distance	Final Distance
1	1822	923
2	1688	1008
3	1885	980
4	1912	1084
5	1909	1054
6	1929	903
7	1683	956
8	1942	1064
9	1991	998
10	793	984
AVG	1855.4	995.4

Table 3: Population 5

Trial	Initial Distance	Final Distance
1	2084	1270
2	2123	1549
3	2210	1518
4	1907	1366
5	2299	1344
6	2270	1479
7	2130	1308
8	2088	1414
9	1991	1325
10	1758	1432
AVG	2086	1400.5

Table 4: Mutation rate .05

Trial	Initial Distance	Final Distance
1	1831	941
2	1852	932
3	1898	1107
4	1914	1053
5	1931	1011
6	1940	1031
7	1797	1020
8	1792	1000
9	1880	906
10	1682	873
AVG	1851.7	987.4

Analysis of Results

The genetic algorithm was run on the TSP ten times for each variation of the algorithm and the results were averaged. From the first table we can see the average initial distance is 1877.1. From this we can only expect the genetic algorithm to go through its evolutions slowly lowering the number as it finds more and more optimal routes. The average final distance is 1037.4, a great improvement on the original route.

For Table 2, the number of evolutions was doubled from 100 to 200. If you look at only the initial distance values, nothing has changed as is expected. The average initial distance was only twenty less than for the default settings. The best initial route is still calculated the same way, the algorithm is simply run more times so that can be accounted from the variation due to randomness inherent in the algorithm. From this we could expect the average final distance to be slightly lower, as 100 evolutions was already on the high end. The average found for when 200 evolutions is used was 995.4, which is less than 50 better. This means that either 100 evolutions is enough to reach a near optimal route and that 200 only marginally selects a more optimal route (not worth it for doubling the computation time), or that the result is not statistically significant considering there were only ten trials run.

For Table 3 the population was reduced from 50 all the way to only 5 individuals. The expected effect is that the initial distance is drastically higher as only 5 routes can be randomly generated and the chances of any good routes appearing in only 5 individuals is extraordinarily low. The average initial distance was found to be 2086, confirming this prediction. After 100 evolutions the average final distance was found to be 1400, much higher than the 1037.4 and 995.4 of previous experiments. This can be explained by the fact that even though there are still 100 evolutions and the mutation rate is still at the default .015, there are only 5 individuals that can crossover between each other so the final distance is heavily dependent on those five routes initially created. The mutation rate is not high enough and the population far too low to produce a route even close to approaching 1000, even through sheer randomness. On the other hand the computation time was lower; this is the tradeoff that is made. Ideally if you ran this experiment with a population of ten thousand, the final distances found would be lower but the computation would take far longer due to the space and time complexity of the genetic algorithm.

For Table 4 the mutation rate was increased from .015 to .05. This increases the amount of shuffling during the mutation phase of the algorithm,

adding some random variation to the routes in between evolutions. My prediction was that increasing the mutation rate would help finding a more optimal route but only to a certain point; at some point the mutation rate is too great and would interfere with almost optimal routes. 0.05 seemed like a reasonable rate to experiment with; the average initial distance was unsurprising at 1851.7 as the mutation rate did not affect this (as with increasing the number of evolutions). The average final distance was 987.4, lower than the 1037.4 with a .015 mutation rate, but not significantly so. Without attempting more trials this would fall in line with my prediction that the final distance improved as we are still in the safe zone of mutation rates, but increasing it to 0.1 or higher could possibly have detrimental effects.

Conclusion

In conclusion, a genetic algorithm is an effective way of finding a good route for a Traveling Salesman Problem. Increasing the number of evolutions undertaken by the algorithm also improves the final optimal route discovered but only marginally so after 100 evolutions; disproportionately increasing the computation size. Decreasing the population has the opposite effect, drastically increasing the cost of the final optimal route discovered. Increasing the mutation rate also benefits the algorithm but only within a certain range. The range was not found in this experiment and cannot be extrapolated without further work. In terms of future work, more trials would absolutely be necessary to be able to ascertain whether the differences between 100 and 200 evolutions is statistically significant. I would encourage 100 trial runs per modification. For the mutation rate I would do 100 trial runs each of .010, .050, .01, .02, and .03 to try and find the sweet spot of mutation rate to promote the most optimal routes.

References

[1] Rego, Csar ; Gamboa, Dorabela ; Glover, Fred ; Osterman, Colin. European Journal of Operational Research, 2011, Vol.211(3), pp.427-441

[2] Wei, Zhen ; Ge, Fangzhen ; Lu, Yang ; Li, Lixiang ; Yang, Yixian. Nonlinear Dynamics, 2011, Vol.65(3), pp.271-281

[3] Zar Chi Su Su Hlaing and May Aye Khine, "Solving Traveling Salesman Problem by Using Improved Ant Colony Optimization Algorithm,"International Journal of Information and Education Technologyvol. 1, no. 5, pp. 404-409, 2011

[4] M. Dorigo and L. M. Gambardella. Ant colonies for the traveling salesman problem. BioSystems, 43:7381, 1997

[5] Xuesong Yan, International Journal of Computer Science Issues, 01 November 2012, Vol.9(6), pp.264-2717

[6] Contreras-Bolton, Carlos ; Parada, Victor. PloS one, 2015, Vol.10(9), pp.e0137724

[7] Karapetyan, D. ; Gutin, G. European Journal of Operational Research, 2011, Vol.208(3), pp.221-232

[8] Erdoan, Gne ; Cordeau, Jean-Franois ; Laporte, Gilbert. European Journal of Operational Research, 2010, Vol.203(1), pp.59-69

[9] Jacobson, Lee. The Project Spot, 2012. http://www.theprojectspot.com/tutorial-post/applying-a-genetic-algorithm-to-the-travelling-salesman-problem/5