Solving the Traveling Salesperson Problem with Particle Swarm

Madison May[1] Caleb Eardley[1] Austin Lyman[1]

Idaho State University, Pocatello, Idaho, U.S.A

Abstract

The travelling salesman problem is one that has been attempted many times before. A particle swarm algorithm is one which uses a large number of potential solutions moving towards one singular good solution to solve it. Our implementation isn't perfect, but our creativity may provide some useful insights for other solutions.

1 Introduction

We chose to solve the Traveling Salesperson Problem (TSP) using a particle swarm optimization algorithm. Flocks of birds and other swarms of organisms in nature inspired particle swarm algorithms. A flock of birds might spread throughout a field to find food, but when one of the birds has more success than the other birds, the others will move closer to successful birds and form a 'swarm'. A particle swarm algorithm mirrors this with a swarm of potential solutions that step towards the best solution.

2 Algorithm Explanation

Our algorithm generates $3n$ random permutations of the list of cities that could be a potential solution. A higher number of particles yields better results, but also slows down the algorithm. After experimentation, we determined $3n$ to be a healthy number of partilces with respect to time and accuracy. Many of these permutations could have an infinite length because they are not a valid path, but that is what we want. It is okay if a particle has infinite length because one step towards the BSSF can change it into a valid path.

The city with the shortest path length is set as the initial BSSF. If at any point a solution is better than the BSSF, it becomes the BSSF. We guarantee that our algorithm will at least be as good as the Greedy solution because we insert the Greedy solution as one of our particles. The algorithm iterates until there is only one remaining solution. At each iteration, each of the particles moves one step closer to the BSSF. They also have a 1/5 chance of performing a local search to find a better solution near them. If they overlap with any other solution, we delete them.

A particle's closeness to the BSSF is defined by how many cities in common it has with the BSSF. For example, a permutation (3, 2, 1, 4) compared with (2, 3, 1, 4) would be considered two 'moves' apart. To step closer, a particle randomly chooses a city it does not have in common with the target particle and swaps its ordering to be a step closer to the target city.

3 Complexity

Due to the stochastic nature of our algorithm, it is hard to define it's complexity. It has a possibility to be infinite due to the randomness of it. If we exclude that possibility by assuming

that it will always work towards a viable solution, we end with a runtime of $O(n^5)$. Starting with the move function, we have a runtime of $O(n^4)$, which is comprised of having to move N particles, and then possibly doing a local search which is $O(n^3)$ for each particle. We do this n times, leaving us with $O(n^5)$.

4 Analysis of Results

	Default		Greedy			Branch and Bound			Particle Swarm		
Cities	Time	Cost	Time	Cost	%of Default	Time	Cost	%of Greedy	Time	Cost	%of Greedy
15	0.00170	20435.4	0.00081	11585.6	0.5669377	3.53271	8842	0.7631887	0.048844	9837.8	0.8491403
30	0.05781	37555.2	0.00235	18725.8	0.4986206	TB	TB	TB	0.440554	15861	0.8470132
60	3.70933	77153.6	0.00610	27748.8	0.3596565	TB	TB	TB	3.056400	24925	0.8982370
100	TB	TB	0.03342	39506.4	TB	TB	TB	TB	18.95791	34643.4	0.8769060
200	TB	TB	0.09040	58594.2	TB	TB	TB	TB	153.0281	54301.4	0.9267367

Our implementation of particle swarm does fairly well. It has a 10% decrease in path length, while the speed increase is minimal. Even at 200 cities, it runs in a minute and a half, and still has a 7.5% decrease in length over the optimal solution. On the 15 city limit, it is almost exactly between the optimal (branch and bound) and greedy solutions. Extrapolating this towards the upper bounds, our solution gets an acceptable solution in the amount of time it takes to run.

5 Future Work

Many particle swarm algorithms have multiple BSSFs so that a multiple swarms can form. This leads to a greater variety in solutions and thus increases accuracy. It also affects complexity significantly and was not something we attempted in our algorithm that could have been a great benefit.

As an additional improvement, we could implement multiprocessing since all of the particles move independently of each other.

To improve both run-time and accuracy, we considered giving the particles a dynamic speed so that particles far from the BSSF could take longer steps towards it while those close to the BSSF could take short steps. We did not implement this in our final algorithm, but could improve it significantly.

Lastly, we use a local search algorithm within our algorithm that runs in $O(n^3)$ time. Due to the nature of our algorithm and the travelling salesperson problem, it was the best solution we could come up with. However, a clever improvement to this particular algorithm would benefit our algorithms complexity significantly.