Computer Systems Research Paper Initial Draft
Using Genetic Algorithms to Optimize the Traveling Salesman Problem
2007-2008

Ryan Honig

May 23, 2008

1 Abstract

My goal is to create a program that can solve the Traveling Salesman Problem, finding near-optimal solutions for any set of points. I will use genetic algorithms to try to find the optimal paths between the points. In the end, after I create a working alorithm that will find near optimal paths, I hope to create a graphic interface that will display the chosen points and the paths through those points as the algorithm runs.

2 Purpose

The main purpose of my project is to develop my own genetic algorithm that can hopefully find close to optimal solutions for the Traveling Salesman Problem. Once this is done I hope to modify the program to work for asymmetric problems and create a user interface that will graphically display the current problem and run the algorithm to find a solution.

This is a good problem to tackle because it is fairly complex and deals both with some complex algorithms and with some higher level math. By finding an efficient and optimal solution to the traveling salesman problem, it can be applied to the larger NP-complete field of optimization problems which can contribute to many fields of study. The TSP has been around for a long time, but more efficient programs for solving the TSPs are still being created. Many different algorithms have been used to attempt to solve TSPs, including heuristics, genetic algorithms, colony based simulations, and brute force. Heuristics are the best for finding 'good', but not optimal, paths fairly quickly, while genetic algorithms take longer but find more optimal paths.

The paper: "New Genetic Local Search Operators for the Traveling Salesman Problem" by Bernd Freisleben and Peter Merz details how a good way to create an algorithm for the Traveling Salesman Problem is to use a basic heuristic to find the initial pool of paths and then use the genetic algorithm on this pool of paths to find a near-optimal solution. I hope to build off of this approach by creating an algorithm that will work for both symmetric and asymmetric TSPs. Another approach that is detailed by Marco Dorigo and Luca Maria Gambardella in "Ant Colonies for the Traveling Salesman Problem" is to use a simulated ant colony to solve a TSP data set. While this is not the most efficient way of solving a TSP, it can find very near-optimal solutions. One of the most interesting articles that I found on the Traveling Salesman Problem is "Genetic Algorithms for the Traveling Salesman Problem: A Review of Representations and Operators". This article does a comparison of the different types of algorithms used to solve TSPs and their different way of representing the data. The question that I would like to answer through my project is what combination of algorithms can create the most efficient and optimal

traveling salesman program.

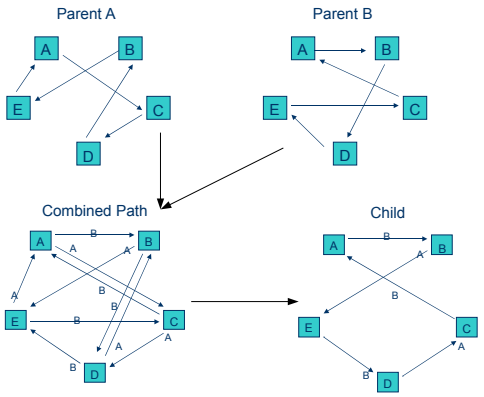
3 Development

3.1 Initial algorithm

¡++¿ With my project, I would like to develop an efficient algorithm that can find near-optimal solutions for both symmetric and asymmetric traveling salesman problems and then incorporate it into a user interface that will run the algorithm and display the paths that the algorithm comes up with. My algorithm will be a mix of basic heuristics and the more complex genetic algorithms.

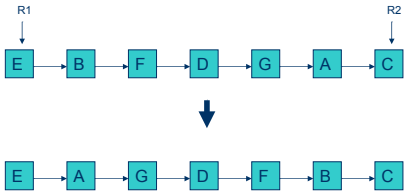
I began by creating a program that used a simple genetic algorithm that would reverse a section of a parent path which would then be replaced in the pool if it had a shorter path than the parent. I began testing this with data sets that can be found here: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/. After finding that my solutions were off by multiple powers of ten, I discarded that algorithm and began a new one.

3.2 Genetic Algorithm



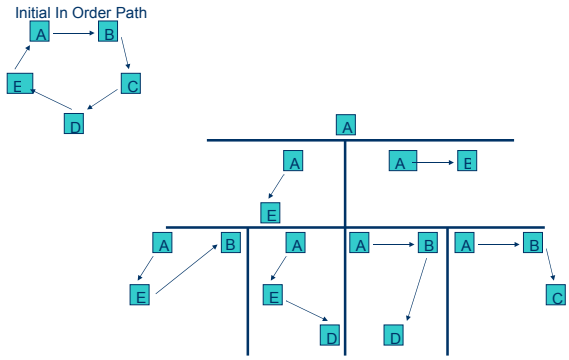
This new algorithm starts by creating an initial pool of fifty random, legal paths. For each iteration of the genetic algorithm it will then select two parent paths at random to create a child path from. All of the links between each point on the parent path are then compiled into one set of links. The program will then alternate choosing a link from each of the two parents to create the crossover. If the program gets stuck on a node and cannot create a legal link from the parent links, then a greedy algorithm takes over and completes the broken path.

3.3 Mutation Algorithm



During second quarter I created a mutation method. This mutation method keeps the pool from being populated by the same path, since it has a chance of changing one of the pools in the path. My mutation method has a one in fifty chance of occuring. When a mutation does occur, two points are selected at random on the path, and then the path in between these two points is reversed. Once my mutation method was implemented, it significantly helped my program because it allowed the pool to continue running even if it got stuck on a single path that wasn't anywhere close to the optimal solution.

3.4 Pool Generating Heuristic



During second quarter, I also created a heuristic to generate the initial pool of paths. I created the heuristic, hoping that it would produce better results by starting with a pool that isn't random and it might even be faster. The heuristic I devised will first pick a random point out of all of the points the salesman must travel to. It then finds which two other points are the closest to that point and begins two paths starting at the first point, and going to each of the other two points. Then, for each of those two points, it finds the next two closest points, and creates two more new paths, thus doubling the number of paths being made. It continues doing this until there are enough points to fill the pool, at which point it will just continue by picking the next closest point, until a full traverse of the points is acheived. I will discuss how this heuristic did in my results section.

4 Results and Discussion

After testing my initial algorithm that reversed sections of the paths, I was not surprised to find that my solutions to data sets were multiple powers of ten off from the best known solutions. I knew that since my initial algorithm was based off of single parent genetics, it would not work very well.

I then created the genetic algorithm that I am currently using. When I first began testing this algorithm, my program would often fill up its pool with copies of the same path, which would prevent it from finding a solution any better than that one. In order to correct this I implemented a mutation method to free up the pool. This worked and my program ran pretty well. Using data set a280 from the TSPLIB website, the best solution that my program came up with was 2608.837612, which has an error of just 1.16 percent from the best known solution of 2579, with an average running time of about 2.15 seconds. Using the att48 data set, my programs best solution was 10820.248365, which has an error of just 1.81 percent from the best known solution of 10628, with an average running time of 3.52 seconds.

I then created my heuristic, hoping that it would produce better results by starting with a pool that isn't random, and possibly even be faster. When testing the heuristic program with the same data sets that I used to test the program with the randomly generated pool, I found that the solutions were slightly better, but the program took mush longer to run. Using data set a280, the best solution that my program came up with was 2597.401845, which has an error of just .72 percent from the best known solution of 2579, with an average running time of about 5.03 seconds. Using the att48 data set, my programs best solution was 10751.542837, which has an error of just 1.16 percent from the best known solution of 10628, with an average running time of 7.31 seconds. Currently, I am not sure whether I should continue working with my heuristic program or with my randomly generated pool program, because although the heuristic program is slightly better, it takes much more time to run.

5 Bibliography

—Dorigo, Marco and Gambardella, Luca Maria. "Ant colonies for the Traveling Salesman Problem". http://code.ulb.ac.be/dbfiles/DorGam1997bio.pdf

—Freisleben, Bernd and Merz, Peter. "New Genetic Local Search Operators for the Traveling Salesman Problem". http://www.rfai.li.univ-tours.fr/pagesperso/rousselle/docum/pdf/ppsn96.pdf

—Larranaga, P., Kuijpers, C.M.H., Murga, R.H., Inza, I., and Dizdarevic, S. "Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators". http://wedhusprucul.tripod.com/skripsi/tsp.pdf

—University of Heidelberg Department of Computer Science. "TSPLIB". http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

—Voudouris, Christos. "Guided Local Search and Its Application to the Traveling Salesman Problem". http://www.cs.essex.ac.uk/CSP/papers/VouTsa-GlsTsP-Ejor98.pdf

6 Appendices

6.1 An Overview of the Traveling Salesman Problem

The Traveling Salesman Problem is a problem in which a set of points is given and you want to find the shortest path that travels between each point once and then returns to the starting point. A symmetric problem is one in which the distance between towns A and B is the same as the distance between towns B and A. An Asymmetric problem is one in which the distance between towns A and B is different from the distance between towns B and A.

6.2 What is a Genetic Algorithm?

A Genetic Algorithm is a process for an algorithm that simulated genetics. First a pool of solutions is generated. Then for each generation of the program that is run, 2 of the solutions in the pool are chosen at random. These two solutions are then somehow combined to create a child solution. A fitness function is then used to determine whether the child solution is better than other solutions in the pool. If it is, then it will replace a solution in the pool. This process continues for many generations, until an optimal solution is found.