Using Genetic Algorithms to Optimize the Traveling Salesman Problem

By: Ryan Honig

Thomas Jefferson High School for Science and Technology Computer Systems Lab

2007-2008

Abstract

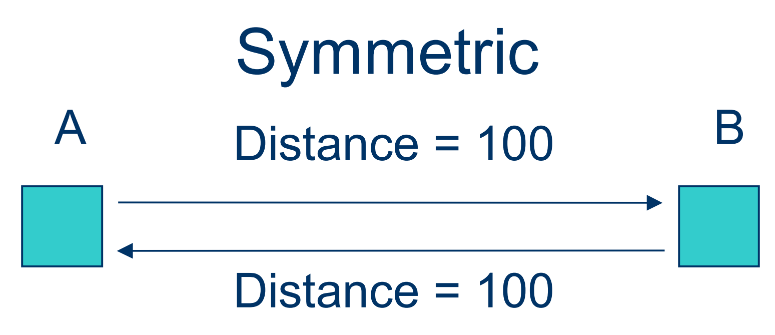
My goal was to create a program that can solve the Traveling Salesman Problem, finding near-optimal solutions for any set of points. I used genetic algorithms to try to find the optimal paths between the points. I compared the program ran when it was using an initial random pool and when it was using an initial heuristically generated pool. I also attempted to modify my program to allow for asymmetric travelling salesman problem input.

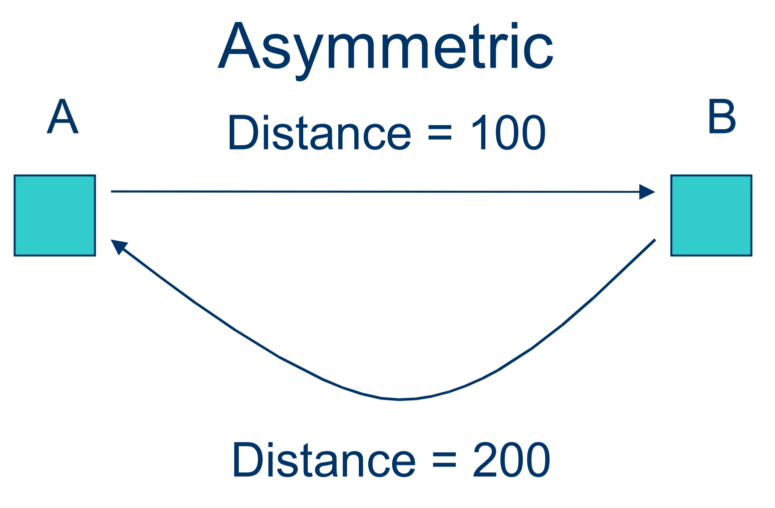
What is the Traveling Salesman Problem

Traveling Salesman Problem (TSP) - a set of points is given. Try to find the shortest path that travels between each point once and returns to the starting point

Symmetric TSP - distance between towns A and B is the same as distance between towns B and A.

Asymmetric TSP - distance between towns A and B is different from distance between towns B and A.





Development

- I have a genetic algorithm that creates a pool, and then uses genetic crossovers within the pool to find the best solution
- I also have a mutation function that has a one in fifty chance of adding a variation into the pool by reversing a segment of a path, this helps to keep the pool from getting filled by copies of the same path
- I also created a heuristic that creates a better pool than the randomized pool, although it runs much slower
- I also attempted to create a program that would solve asymmetric travelling salesman problems. This proved to be very difficult and I was not able to get it to work in the end.

Future Research

If anyone wants to continue working in this field, they can:
- Finish creating a program that finds solutions to the asymmetric travelling salesman problems
- Create a User Interface to visualize the paths as they are created

Results

Testing the random-pool program against the Heuristically generated pool program

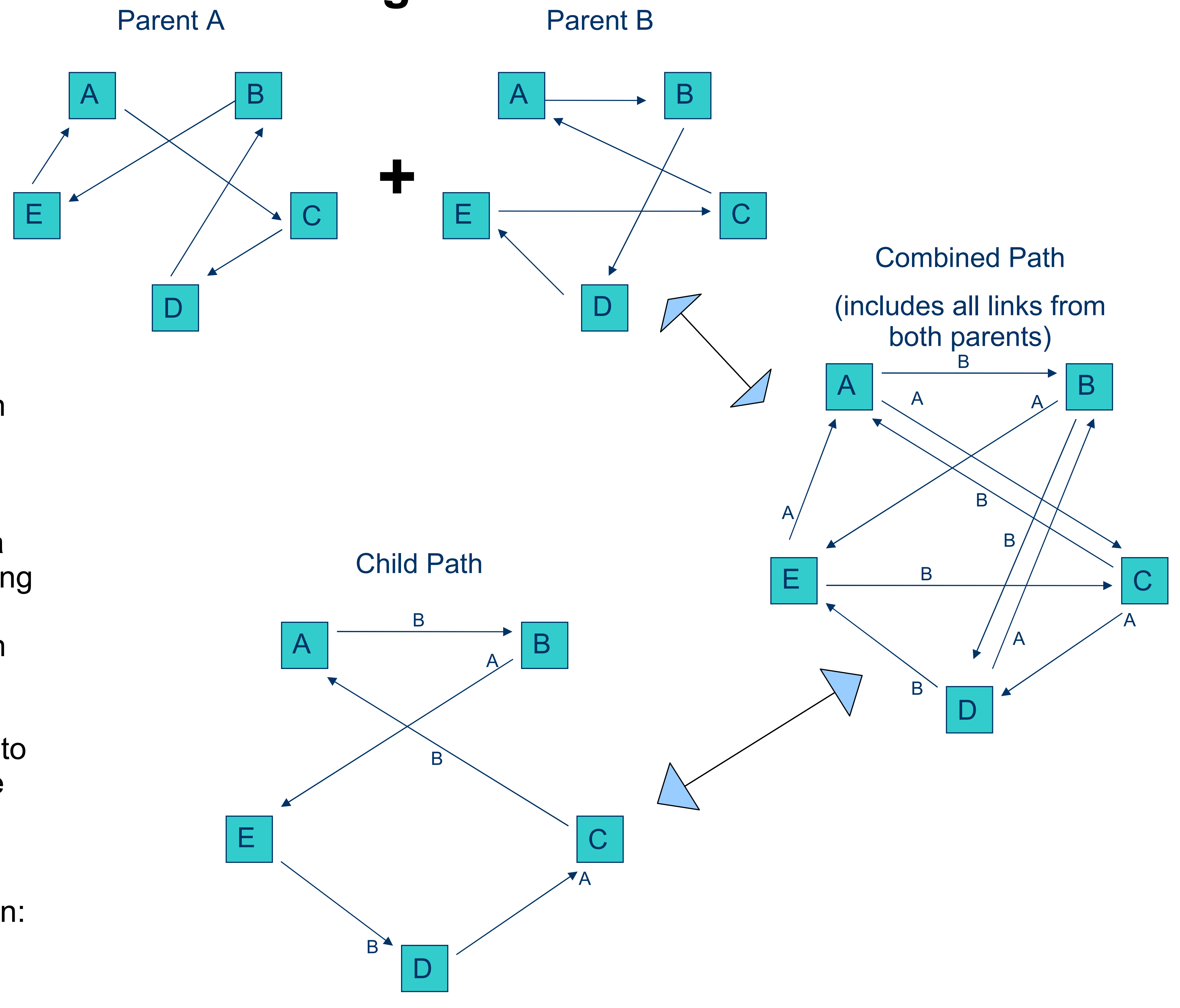
	Random Pool Program		Heuristic Program	
Data Set / Best solution	Average (of 5 runs)	Average Run time	Average (of 5 runs)	Average Run Time
A280: 2579	2780.54	1.75 sec	2729.37	5.11 sec
ATT48: 10628	12017.46	2.31 sec	12104.32	7.32 sec
BAYG29: 1610	1750.92	1.33 sec	1693.84	4.32 sec
BAYS29: 2020	2385.34	1.86 sec	2327.77	5.76 sec
CH130: 6110	6493.65	2.76 sec	6487.37	6.43 sec

As you can see from my data, while the heuristically-generated pool program found slightly better solutions on most of the data sets, with the exception of data set ATT48, on every case it took at least twice as much time to run than the randomly generated pool program did. In the end, I concluded that the benefit in solution length from the heuristic program was not enough to justify the long run times.

Background

- Purely genetic approaches can find near optimal solutions, but have a long run time
- Purely heuristic approaches can run very efficiently, but don't find very optimal solutions
- Many of the current best known solution algorithms use a combination of heuristics and genetic algorithms

How My Genetic Algorithm Works


Parent A
A
B
C
D
E
+
Parent B
A
B
C
D
E
Combined Path
(includes all links from both parents)
A
B
C
D
E
Child Path
A
B
C
D
E