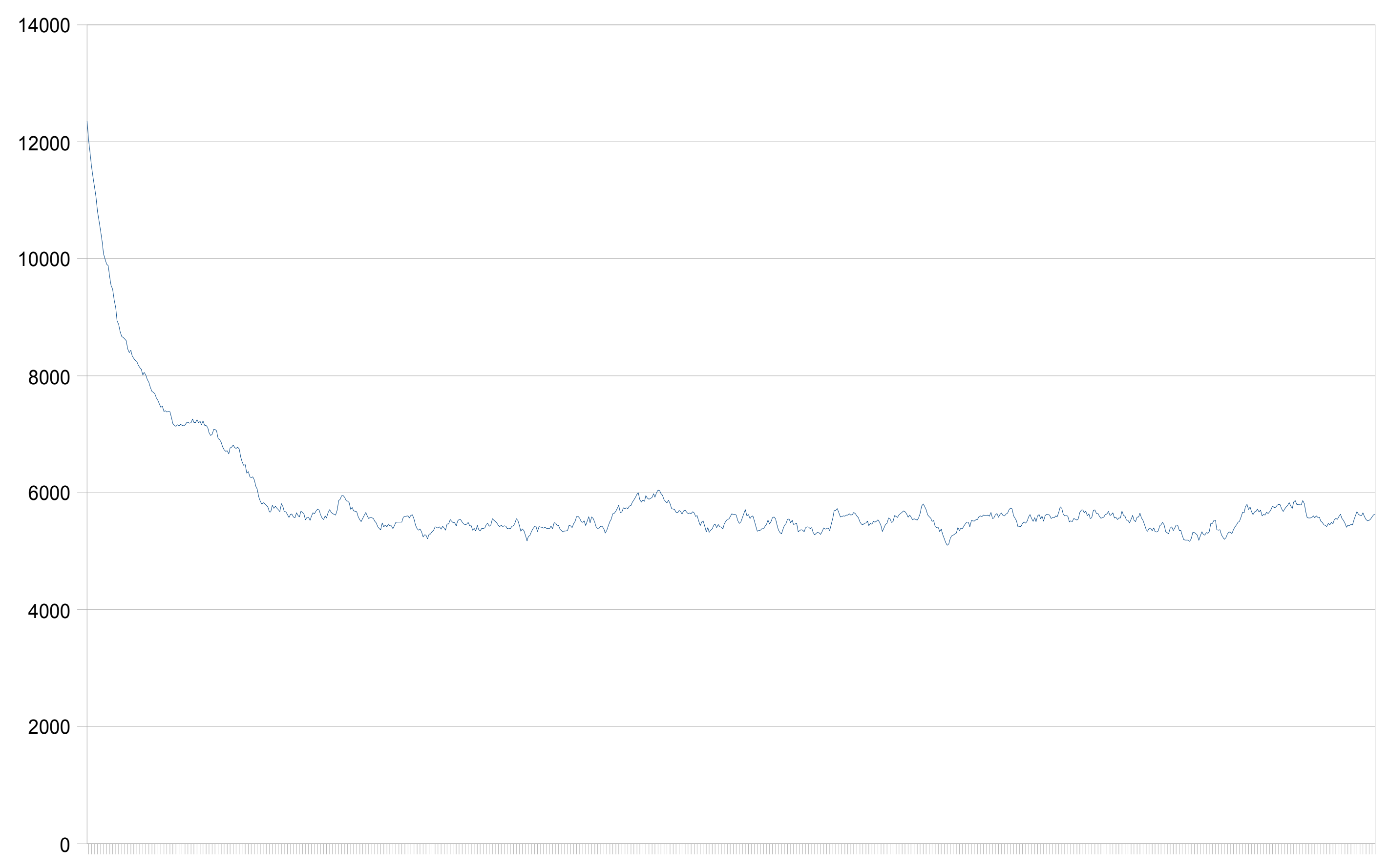
Genetic Algorithms to find Near Optimal Solutions to the Traveling Salesman Problem(TSP)

The Traveling Salesman Problem (TSP) is the classic nondeterministic polynomial-time hard(NP-hard) problem. The problem goes as such Given a number of cities and the costs of traveling from any city to any other city, what is the cheapest round-trip route that visits each city exactly once and then returns to the starting city? Although the problem is stated so simply and discreetly it is in fact a very difficult problem to solve. To simply use brute force to check all possible solutions it would require a O(N!). This quickly becomes very difficult to do even for relatively small n. Therefor it becomes pertinent to find near optimal solutions through other methods using more realistic times.

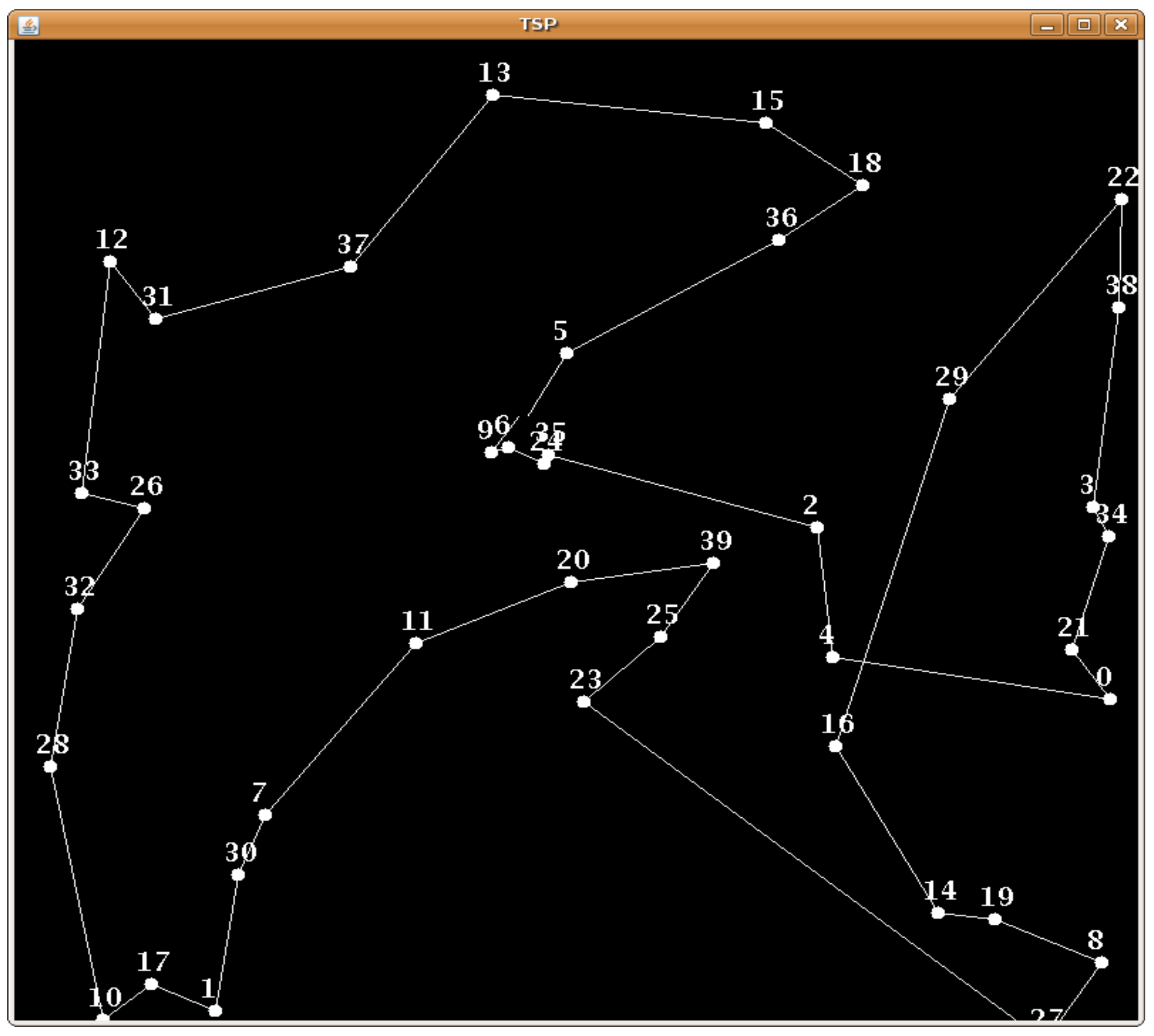
The methods which I use to find near optimal solutions to the TSP are genetic algorithms(GA). GAs are a search technique in computing used for optimization and search problems such as the TSP. GAs are inspired by evolutionary biology and incorporate such concepts as inheritance, mutation, selection, crossover, and reproduction.

```
Pseudo-code for generic GA
1)Choose initial population
2)Evaluate fitness for each individual in the population
3)Repeat
    1)Select the best individuals to reproduce
    2)Breed new generations using crossover and mutation
    3)Evaluate fitness of the offspring
    4)Replace worst ranked part of population with offspring
```

Fitness Level Verses Time



Sample TSP



Another method in which I will be employing in order to solve the TSP will be through Ant Colony Optimization. In ant colony optimization I use simulated ants which tour through the search space and leave a pheromone trail. Based on how efficient the trail is more or less of the pheromone is evaporated. Ants then stoicastically chose the various trails in order to find a near optimal solution.

An ant will move from node I to j with probabiliity -

$$pi,j = \frac{[\tau ij]\alpha[\eta ij]\beta}{\Sigma[\tau ij]\alpha[\eta ij]\beta}$$

```
Psuedo-Code for ACO
 procedure ACO_MetaHeuristic
   while(not_termination)
      generateSolutions()
      pheromoneUpdate()
      daemonActions()
   end while
 end procedure
```

Currently I am using a cycle representation of a solution. This means that I represent each solution as the order in which it attends the cities. This is not very efficient, but it was fairly easy to code.
I am currently using single point mutation and double point crossover. This is how my solutions get better. Eventually I should work for a double point mutation. If I use a matrix encoding the double point crossover will be replaced by a more efficient matrix crossover.
To assess fitness I am currently using the difference between an individual solution and the worst solution in the current gene-pool. This ensures that the worst solution in any given gene-pool does not reproduce. Eventually I want the fitness be placed on an exponential curve in order for the better solutions to reproduce more often.