CS 205
Travelling Salesman Problem using Evolutionary Programming

The travelling salesman problem (TSP) can be stated as:
A salesman is required to visit n cities where each city must be visited exactly once before he returns to the starting city. If the cost is proportional to the distance between cities, what is the least-cost route the salesman can take.

This problem has been worked on since at least the 1800s with a general form of the problem being worked on at length by mathematicians at Harvard in the 1930s.

A brute force solution would require that we determine the total distance of all possible routes. Say, for example that there are just five cities. If the salesman starts in city 1 then he has four choices for his first trip, three for his second, two for his third, and just one choice for his fourth trip which gets him back to his starting point. To generalize, if there are n cities there are (n-1)! possible paths. If n is small, this problem is solvable in an optimal way but as n grows n! grows very rapidly and for a large number of cities the brute force solution is computationally impossible even for the fastest super computer. Consider that $99! = 9.332 \times 10^{155}$.

We can arrive at a reasonably good approximation to the optimal solution using evolutionary programming which works like this:

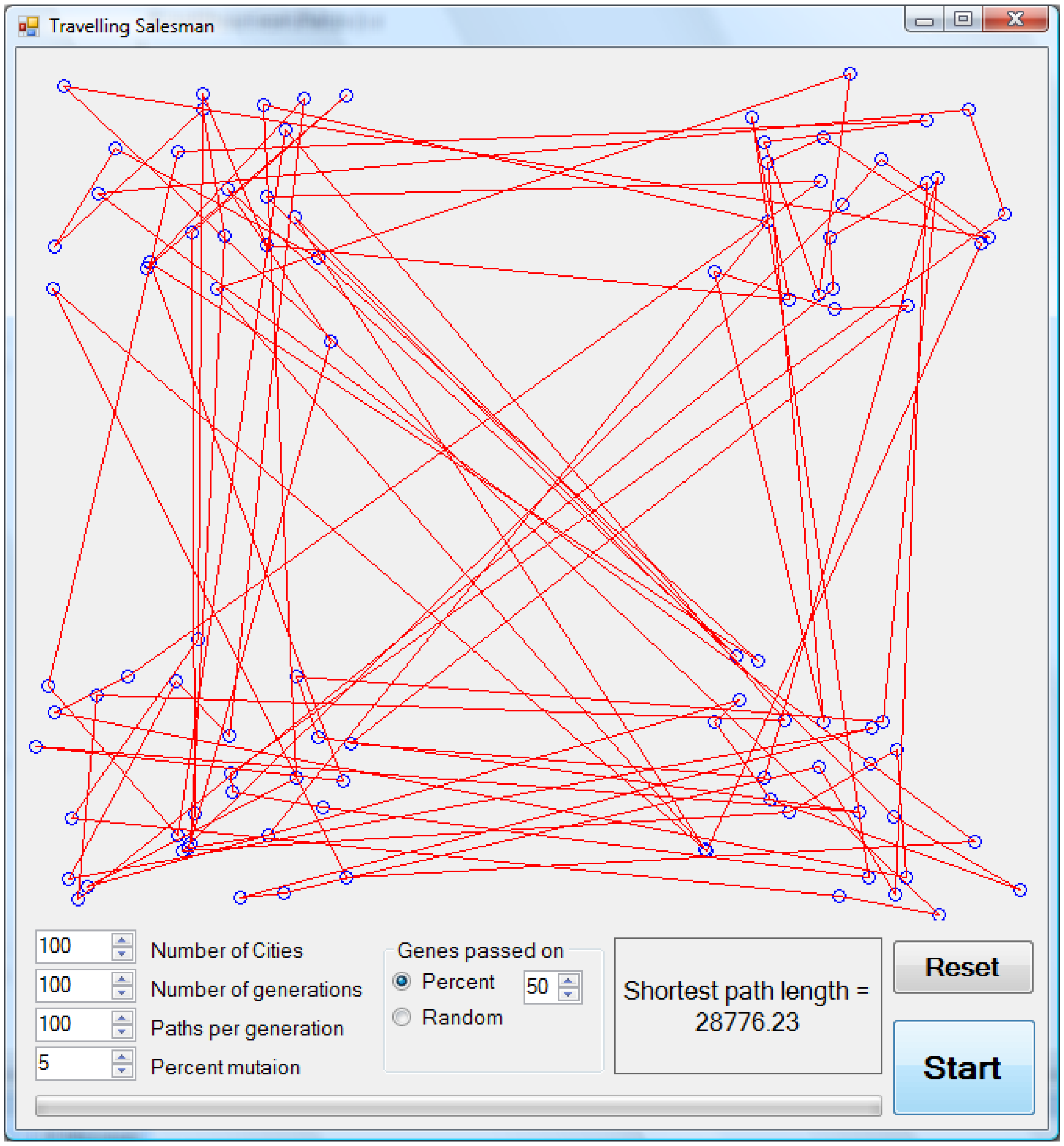
- If we have say 100 cities there are 99 distances we have to calculate for each path. Generate say 1000 random paths, find the total distance of each path, and sort them with the shortest path first. Call these 1000 random paths generation 0.
- For the next generation use shortest path to generate 999 new paths by taking a random subsection of the shortest path and adding onto it a random walk through the remaining cities. For example, if there were only five cities and say the shortest path after generation 0 is {3, 5, 2, 4, 2}. Choose a random section from this path say {5, 2, 4} and randomly select the remaining cities so that the new child path could be {5, 2, 4, 3, 2}. Sort all of the paths and repeat this process for k-generations.

Class exercise:
A program which does this and uses graphics to display the shortest path can be downloaded from the web site as TravellingSalesman.zip. Download the program, unzip it and verify that it runs.

A typical run of the Travelling Salesman program will produce a screen similar to that shown in Figure 1. The user can enter the following options:

- Number of Cities – how many cities will the salesman visit. Can be 3 to 10,000.
- Number of generations – how many generations to run. Varies from 1 to 10,000.
- Number of paths per generation – how many paths to try for each generation. Varies from 1 to 10,000.
- Number of mutations – how many paths (as a percentage) undergo mutation in each generation.

- Genes passed on – Can be random or a percentage. If it is random then a path of random length is chosen from the parent string and copied to each child string. If it is not random then the user can specify the amount of the parent's path that gets passed to each child. This can vary from 0% to 99%.



100 Number of Cities
100 Number of generations
100 Paths per generation
5 Percent mutaion

Genes passed on
[✓ Percent] 50
Random

Shortest path length = 28776.23

Figure 1

A typical run of the Travelling Salesman program. The blue circles represent cities and the red lines are the paths between the cities taken by the salesman.

This version of the program has two classes which are the *City class* and the *Path class*. The City class has an x and a y location for the city and a Boolean variable called *visited* which is true if the city has been visited by the salesman. This class implements properties X, Y, and Visited to allow access to these variables.

The Path class has just two private variables which keep track of the path and its length. The path is stored in an int array called p. Each element of p has an index to a city. The path length is stored as a double. Both p and the path length are available as properties.

Class project: Travelling Salesman project
Download the C# project file *TravellingSalesmanInClass.zip* from the web site. Unzip this file, load it into Visual Studio and verify that it works. Travelling Salesman uses only crossover and mutation. Growth has not been implemented. We are going to modify the mutation, paint, and other methods related to the two classes.

1. The major calculation that has to be done for each new path is to calculate the path length. Our imaginary cities are all in a plane so we use the distance formula given by:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

This calculation is done in the SetPathLength method by the equation given by:

```
 totalDistance += Math.Sqrt((c[cIndex].X - c[cNext].X)*(c[cIndex].X - c[cNext].X)
                      + (c[cIndex].Y - c[cNext].Y)*(c[cIndex].Y - c[cNext].Y));
```

We can make this look prettier by overloading the minus operator in the City class. Doing so would allow us to write

```
        totalDistance += c[cIndex] - c[cNext];     //City class is overloaded
```

Modify the equation for the total distance in the SetPathLength method and add the overloaded minus operator to the City class as a public static method that returns a double.

2. All of the cities in this implementation are the same and are represented with a small blue circle. In the paint method for the panel, make a modification that represents the *first city* as a filled in circle of some other color. To do this you can use the FillEllipse method of the graphics class. FillEllipse requires a brush instead of a pen. You can create a brush like this:

```
        Brush greenBrush = new SolidBrush(Color.Green);
```

This filled in circle of a different color will represent the starting and ending point for the salesman.

3. The Mutation method chooses a random path and a random city on that path along with city the salesman goes to next. It interchanges these two cities. So, for example, if there are five cities and they are visited as {1, 3, 4, 2, 5} and the second city is chosen for mutation on this path the new path will be {1, 4, 3, 2, 5}. Modify this mutation routine so that the two cities are chosen completely at random rather than the two that are adjacent to each other on the path.

4. When the program runs it prints the shortest path length in the status window. Write a new method called private double FindLongestPath() which returns the value of the longest path taken. Use the paint method to print this value in the status box along with the shortest path information. FindLongestPath will be very similar to FindShortestPath except that in this case you need only keep track of the longest length. FindShortestPath also stores the path itself for display.