**Comp 6776 – Assignment 2 – Travelling Salesman Problem**

**Aswini Naveen || 201692818 || anaveen@mun.ca**

**Table:**

|  |  |
| --- | --- |
| * 1. **Initialization method** | Permutation, Random |
| * 1. **Parent selection method** | Roulette Wheel selection biased towards fitness |
| * 1. **Mutation method** | Swap Mutation |
| * 1. **Crossover method** | “Cut and CrossFill” CrossOver |
| * 1. **Survivor selection method** | Replace worst |
| * 1. **Termination condition** | Solution or 10,000 fitness evaluations |
| * 1. **Set of parameters** | 100 – population, 0.3 – Crossover, 0.01 – mutation (recommend lower mutation rate) |
| * 1. **Runtime for each TSP instance** | 1000 generations/minute |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Order of Operations:**

1. Start with a population of potential solutions (routes)- based on population input generate random orders of city’s (Fisher Yates Shuffle is adapted here to generate random order)
2. Select two individuals- element of randomness, but the selection is biased toward fittest:(Roulette wheel selection implemented)
3. Decide if they will exchange DNA(based on crossover rate obtained from frontend), if not pass one of the parents clone to next step (50% chance for both parents).
4. Apply mutation to each pass over Order based on mutation rate logic. Ex., if mutation rate = 1, then all the orders will go through mutation.
5. Repeat 1-2 until we have new population of same size as old one.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**About the Representation in Front End:**

In the case of my index.html, I will be displaying:

1. Fittest individual (route) in our current generation

2. Fittest individual ever produced by this run of the algorithm in this TSP instance.

3. Plot current generations best, worst and average along with the best one ever produced by this run of the algorithm in this TSP Instance.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Scenarios Tested: (w.r.t Standard input to determine best rates):**

Population: GENETIC DIVERSITY

**population 10, cross .3, mutation 0** Generations to achieve best 11783 // **bad, low diversity, takes a lot of generations**

**population 100, cross .3, mutation 0** Generations to achieve best 1629 // **good, high genetic diversity**

Crossover:

**population 50, cross 0, mutation 0** // **bad, local max problem**

**population 50, cross 0.1, mutation 0** // **bad, local max problem, sometimes luckily we get best for standard input**

**population 50, cross .5, mutation 0** Generations to achieve best 686 // **Decent solver**

**population 50, cross 1, mutation 0** Generations to achieve best 121 // **terrible solver**

Mutation:

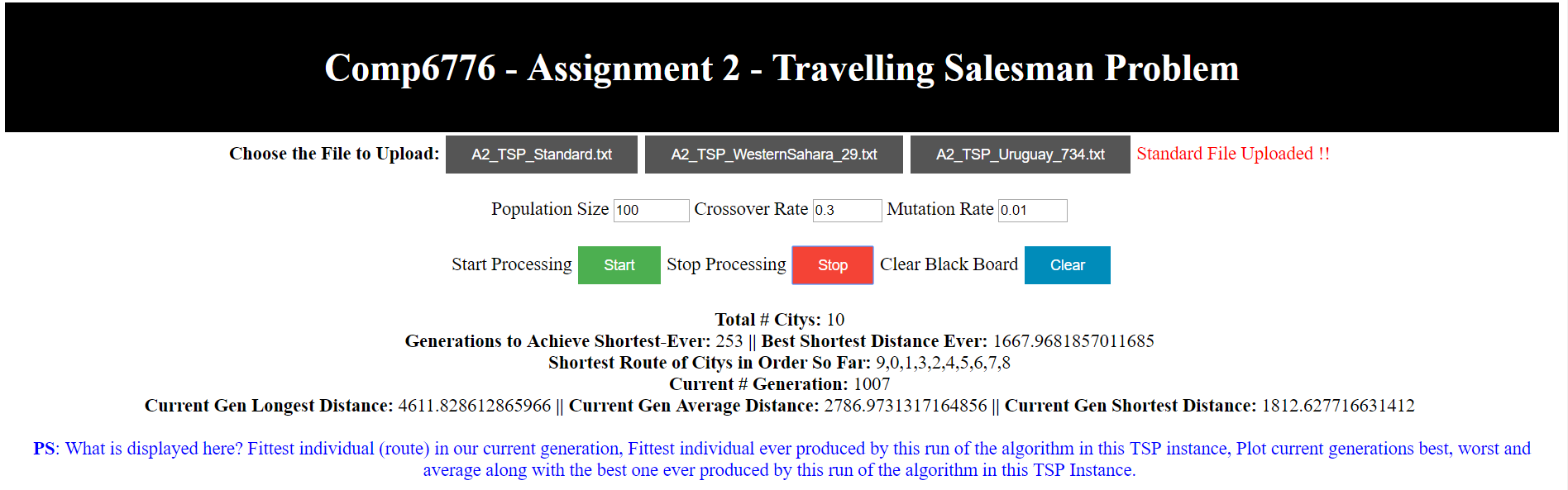
**population 100, cross .3, mutation 0.01** Generations to achieve best 117 // **a terrible solver**

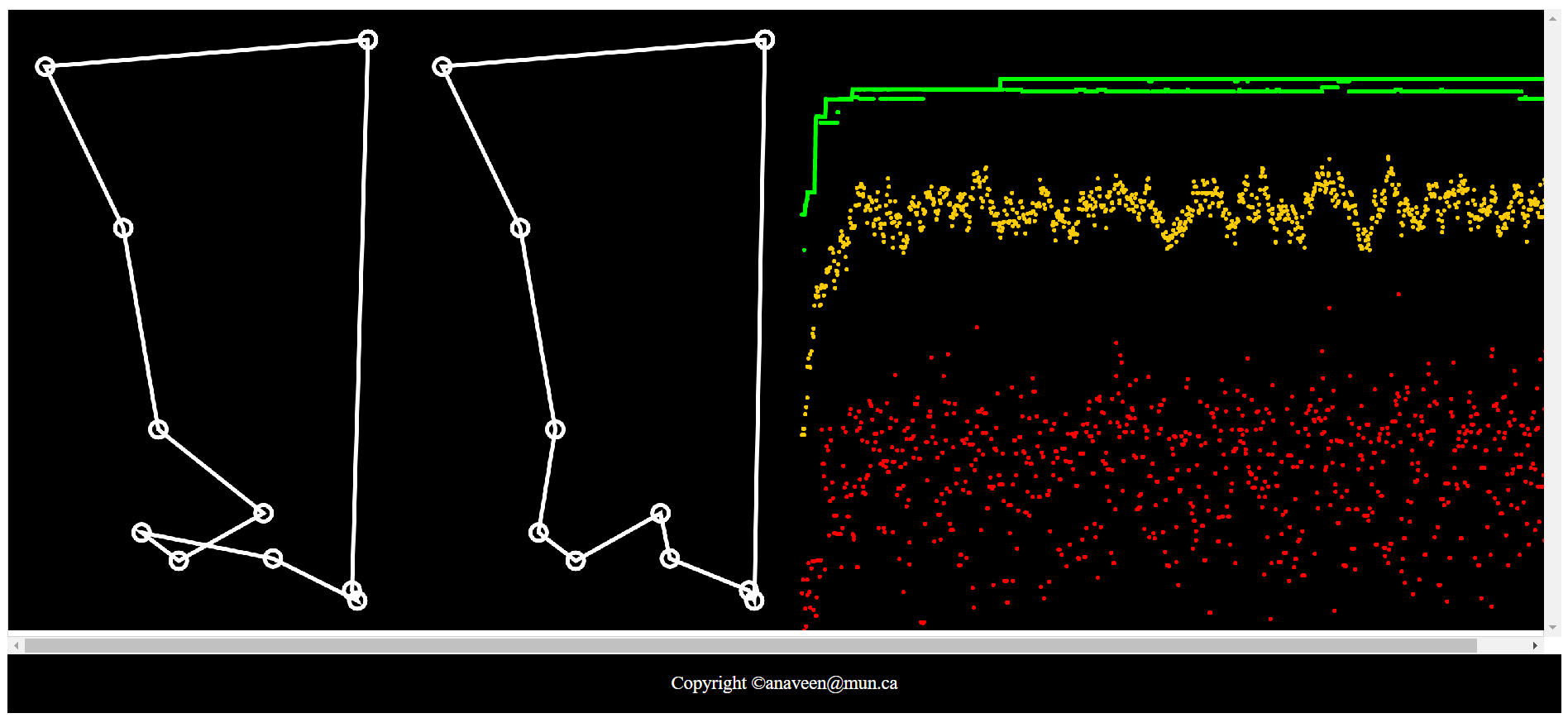
**population 100, cross .3, mutation 1** Generations to achieve best 397 // **very nice solver**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

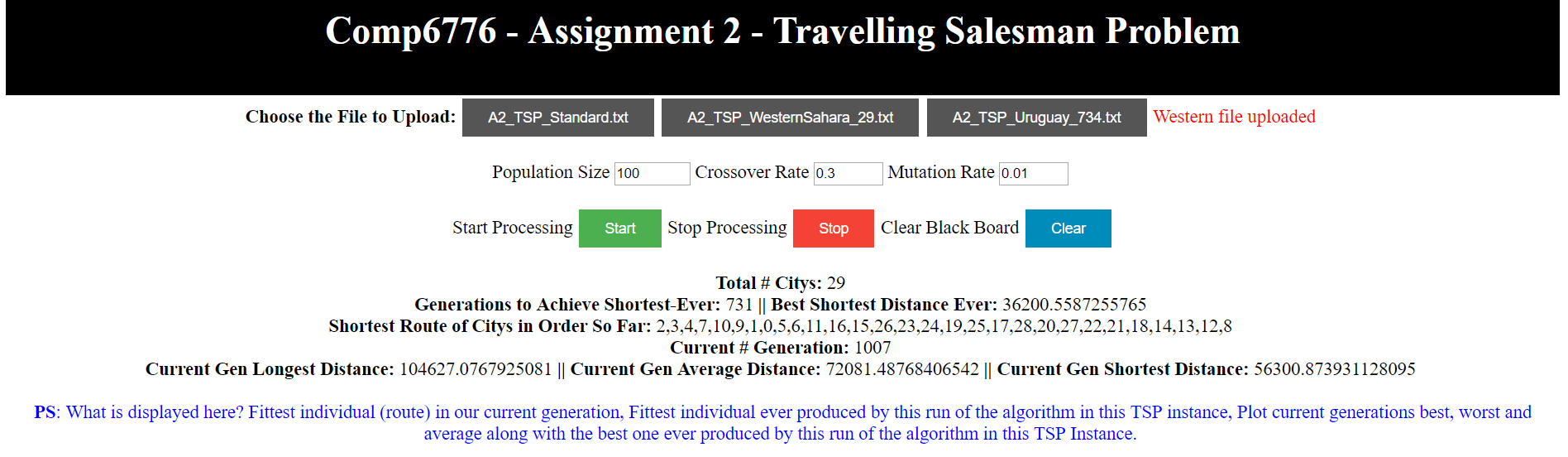
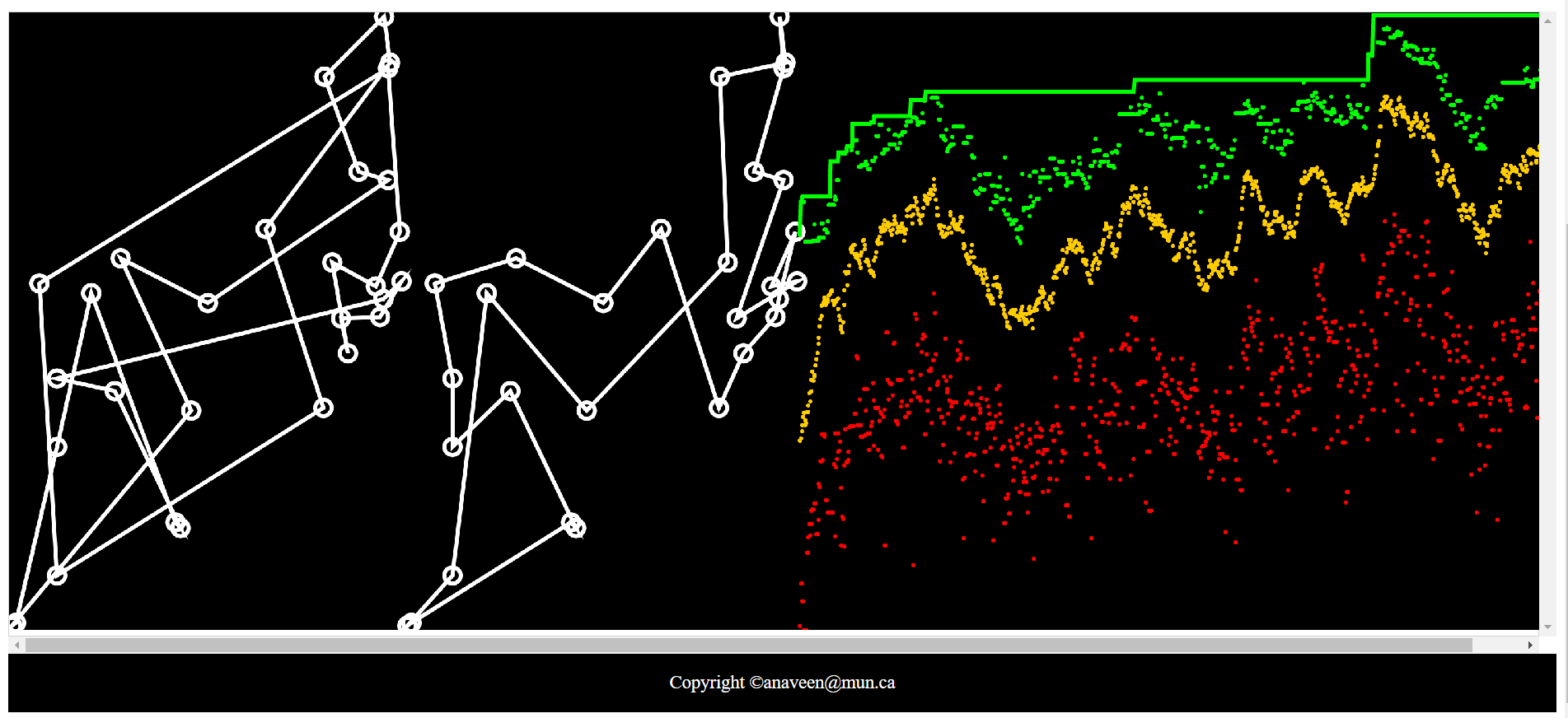
**Screenshots from Actual Implementation: 3 Graphs Attached**

Standard Input File:





**Graph – 1 – Western Sahara – 29 cities**

**Graph 2 – Uruguay 720 Cities**

