**COSC3P71 Assignment 2 general program overview**

This program is written in Java, the program was compiled using Java 1.7 on Windows and OSX.

The program starts off by reading in two files the “berlin52.tsp” and “Parameter.txt.” Then it parses those files to be use by he genetic algorithm.

The Chromosome in my program holds all the cities. I made my chromosome for the GA as and arraylist because it will be easier to add and search for the different cities using the build in methods and function. The “population class” holds an array of “Chromosomes.”

Then my “geneticAlgorithm.class” process the data depending on the parameters data.

***More technical detail on how my program works, I added a lot of comments and detail in my code.***

*Depending on how many times the user want it to run, it takes the average of those runs, after it finishes it, meaning*the genetic algorithm method is done, the program saves the result to “./output/GA\_Test\_Run.txt”

***Parameters Used for Testing***

The genetic algorithm program used a variety of parameters to allow the user to test its functionality. The program reads in a file with multiple parameters for testing such as the following.

**Population size:** Size of the maintained populations. Default chosen: 50.

**Tournament size:** Number of randomly selected individuals in each tournament selection. Default chosen: 3.

**Crossover rate:** Percentage of individuals that crossover should be applied to.

**Mutation rate:** Percentage of individuals that mutation should be applied to.

**Generations size:** The number of times on how many generations the algorithm iterate for. Default chosen: 10000.

**Cross Over Types:** The genetic algorithm program uses two crossover type, **Order CrossOver** and **Uniform Order CrossOver.**

**Order CrossOver:** The program randomly generates two numbers from 1 to the chromosomes size, and have that as the subset to be crossover from parent1 to the children while preserve the relative order in the chromosome from parent one. The rest is copied from parent2 and relative order as well to created a better offspring.

**Uniform Order CrossOver:** A random bit mask of 1 and 0 is generated before the crossover, the child will inherence from parent1 if the bit in the mask is 1, otherwise if it’s a zero the child will inherence from parent2. Before it inherence from parent2 the program checks if that city is not in the chromosome already to a ensure valid offspring chromosomes for the TSP, then it adds it to make sure there are no duplicate.

**Elitism:** elitismis set turn on by default, it keeps the best fitness chromosome from the last generation and adds it to the next generation.

**Elitism Rate:** Same idea as elitism, If elitism rate is turn on, it takes the best percentage from the last generation, if it’s set to 10%, it will keep the best 10% from the last generation and add it to the next generation. **In my entire test I set the elitism rate to 10%**

**Seed:** a random seed value is pass to the random class to ensure a different seed when running multiple runs. Default is 7 if its one run.

In my program the seed value is set to the current iteration count in my number of runs to ensure a different seed every time.

**Number Per Run:** The number of time the GA will iterate for and calculated the average of the fitness and average population fitness.

**Mutator:** The mutation I used in my assignment is **“Insertion”,** it selects a city and swaps it in a random place in the chromosome.

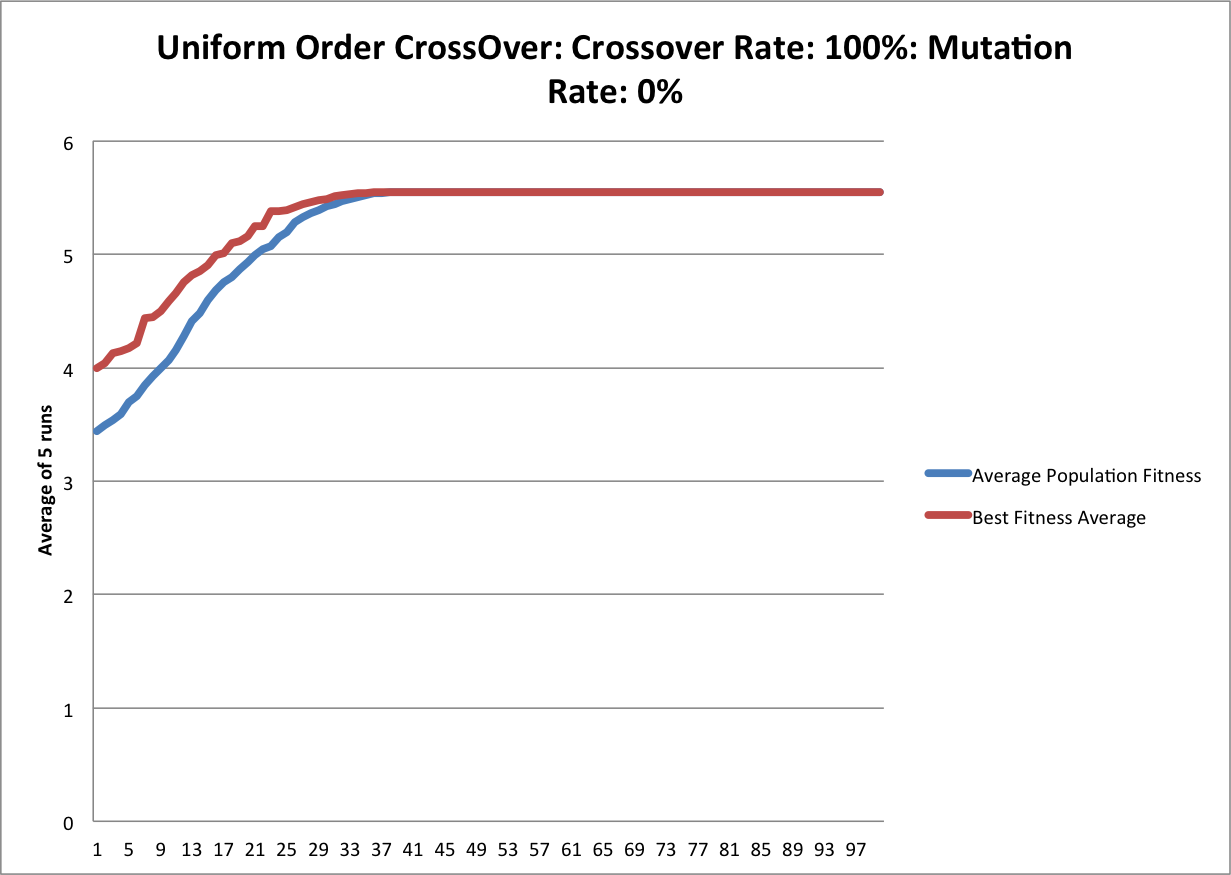
**Objective and Problem Definition**

The objective of this assignment was to create a genetic algorithm that finds an optimal solution for the Travelling salesman. The fitness value was assigned based on the shortness distance between each cities in the given order that chromosome holds when calculating. The smaller that distance value is the higher the fitness value is.

**Test Results**

As the genetic algorithm attempt to find a better fitness solution (shorter distance), it required more generations to achieve. In my case, the algorithm never reaches the solution if the maturation rate is turn off. However, several observations can be made about performance from the data and graphs collected.

The graphs show the average between all five runs for each parameters used. The generation size I used was 10,000 for each test. In the case where nothing really happens after 2000 generation, I just graphs the first 100-generation because the rest of 9900 generation is the same, it gets stuck in a local max.



**Parameter: 1**

numberPerFun: 5

elitism: TRUE

elitismRate: 0.1

crossOver: 1.0

mutationRate: 0

generations: 10000

population: 50

UniformOrderCrossOver: TRUE

OrderCrossOver: FALSE

tournamentSize: 3

**Best of each run:**

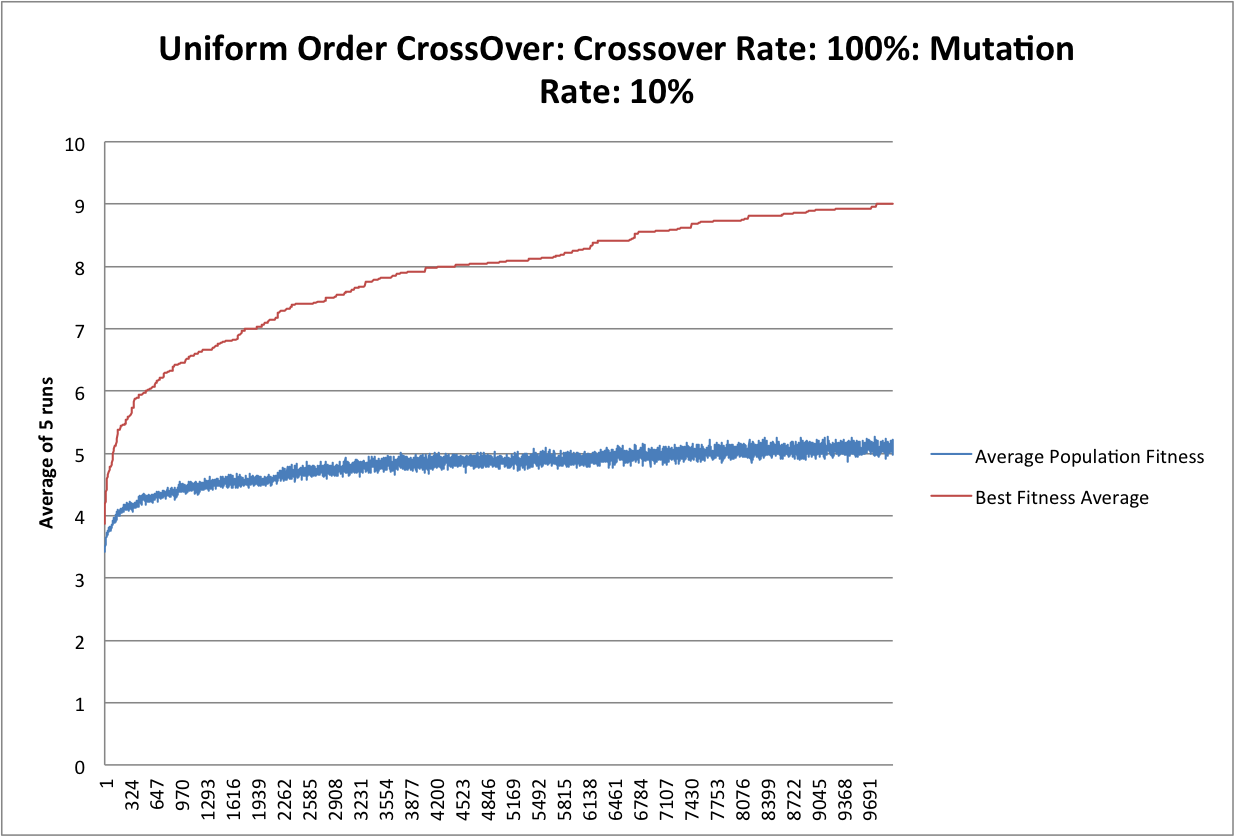
Run: 1 Seed: 0 AverageFitness: 4.907734589713386 Best Fitness: 4.907734589713389 Distance: 20376.0

Run: 2 Seed: 1 AverageFitness: 5.897617362585516 Best Fitness: 5.897617362585516 Distance: 16956.0

Run: 3 Seed: 2 AverageFitness: 5.009769049646815 Best Fitness: 5.0097690496468115 Distance: 19961.0

Run: 4 Seed: 3 AverageFitness: 5.87924040214005 Best Fitness: 5.879240402140043 Distance: 17009.0

Run: 5 Seed: 4 AverageFitness: 6.063178318074335 Best Fitness: 6.063178318074335 Distance: 16493.0



**Parameter: 2**

numberPerFun: 5

elitism: TRUE

elitismRate: 0.1

crossOver: 1

mutationRate: 0.1

generations: 10000

population: 50

UniformOrderCrossOver: TRUE

OrderCrossOver: FALSE

tournamentSize: 3

**Best of each run:**

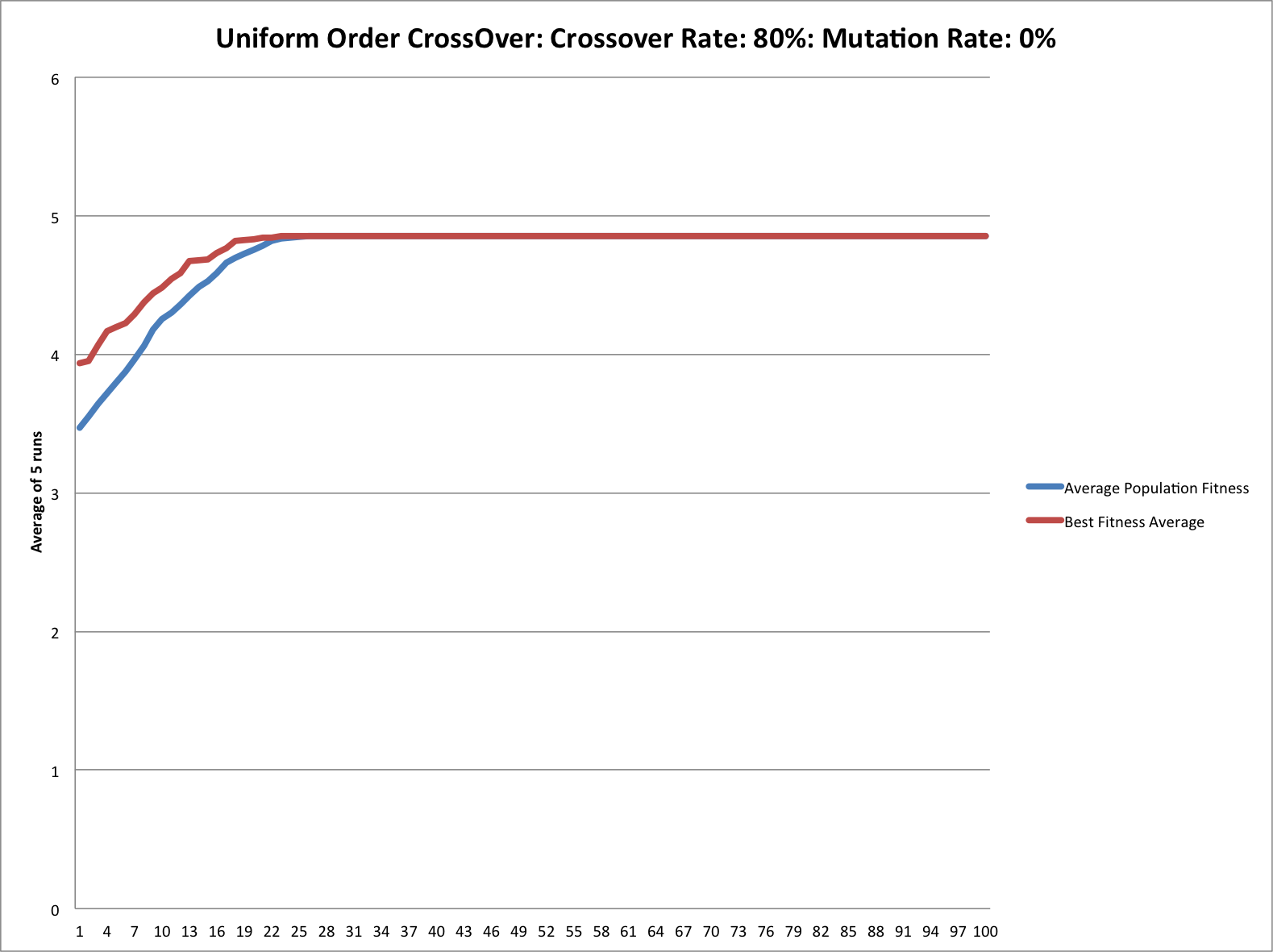
Run: 1 Seed: 0 AverageFitness: 5.0357178001831615 Best Fitness: 9.391435011269722 Distance: 10648.0

Run: 2 Seed: 1 AverageFitness: 4.91459191494279 Best Fitness: 8.58000858000858 Distance: 11655.0

Run: 3 Seed: 2 AverageFitness: 5.225662608405693 Best Fitness: 9.523809523809524 Distance: 10500.0

Run: 4 Seed: 3 AverageFitness: 5.041158670319558 Best Fitness: 8.875477056891809 Distance: 11267.0

Run: 5 Seed: 4 AverageFitness: 4.839913148238383 Best Fitness: 8.664760419374405 Distance: 11541.0



**Parameter: 3**

numberPerFun: 5

elitism: TRUE

elitismRate: 0.1

crossOver: 0.8

mutationRate: 0.0

generations: 10000

population: 50

UniformOrderCrossOver: TRUE

OrderCrossOver: FALSE

tournamentSize: 3

**Best of each run:**

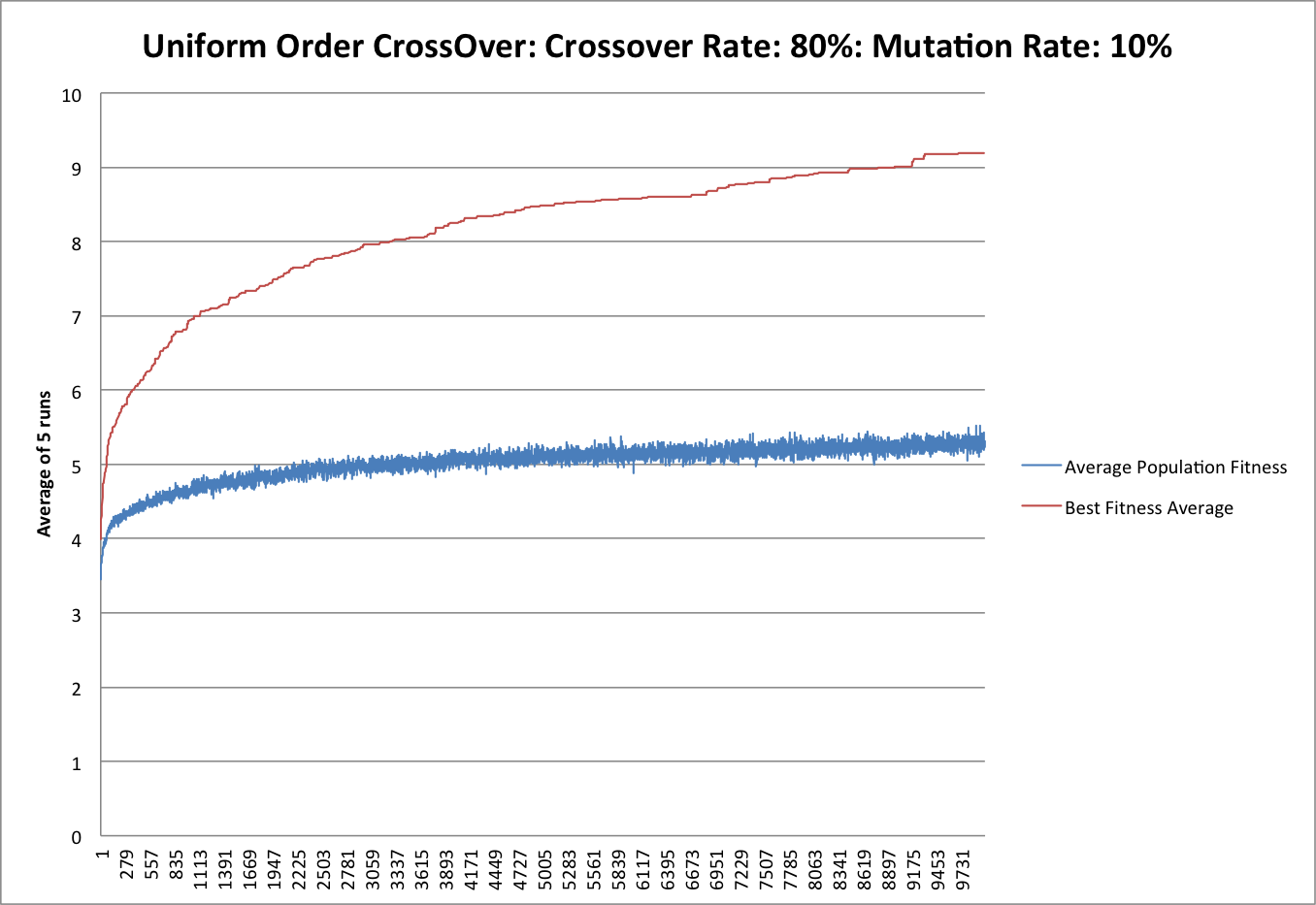
Run: 1 Seed: 0 AverageFitness: 4.495796430337634 Best Fitness: 4.495796430337634 Distance: 22243.0

Run: 2 Seed: 1 AverageFitness: 4.756921320521364 Best Fitness: 4.756921320521359 Distance: 21022.0

Run: 3 Seed: 2 AverageFitness: 5.355325871579283 Best Fitness: 5.355325871579286 Distance: 18673.0

Run: 4 Seed: 3 AverageFitness: 5.149595756733101 Best Fitness: 5.149595756733096 Distance: 19419.0

Run: 5 Seed: 4 AverageFitness: 4.517936206740758 Best Fitness: 4.51793620674076 Distance: 22134.0



**Parameter: 4**

numberPerFun: 5

elitism: TRUE

elitismRate: 0.1

crossOver: 0.8

mutationRate: 0.1

generations: 10000

population: 50

UniformOrderCrossOver: TRUE

OrderCrossOver: FALSE

tournamentSize: 3

**Best of each run:**

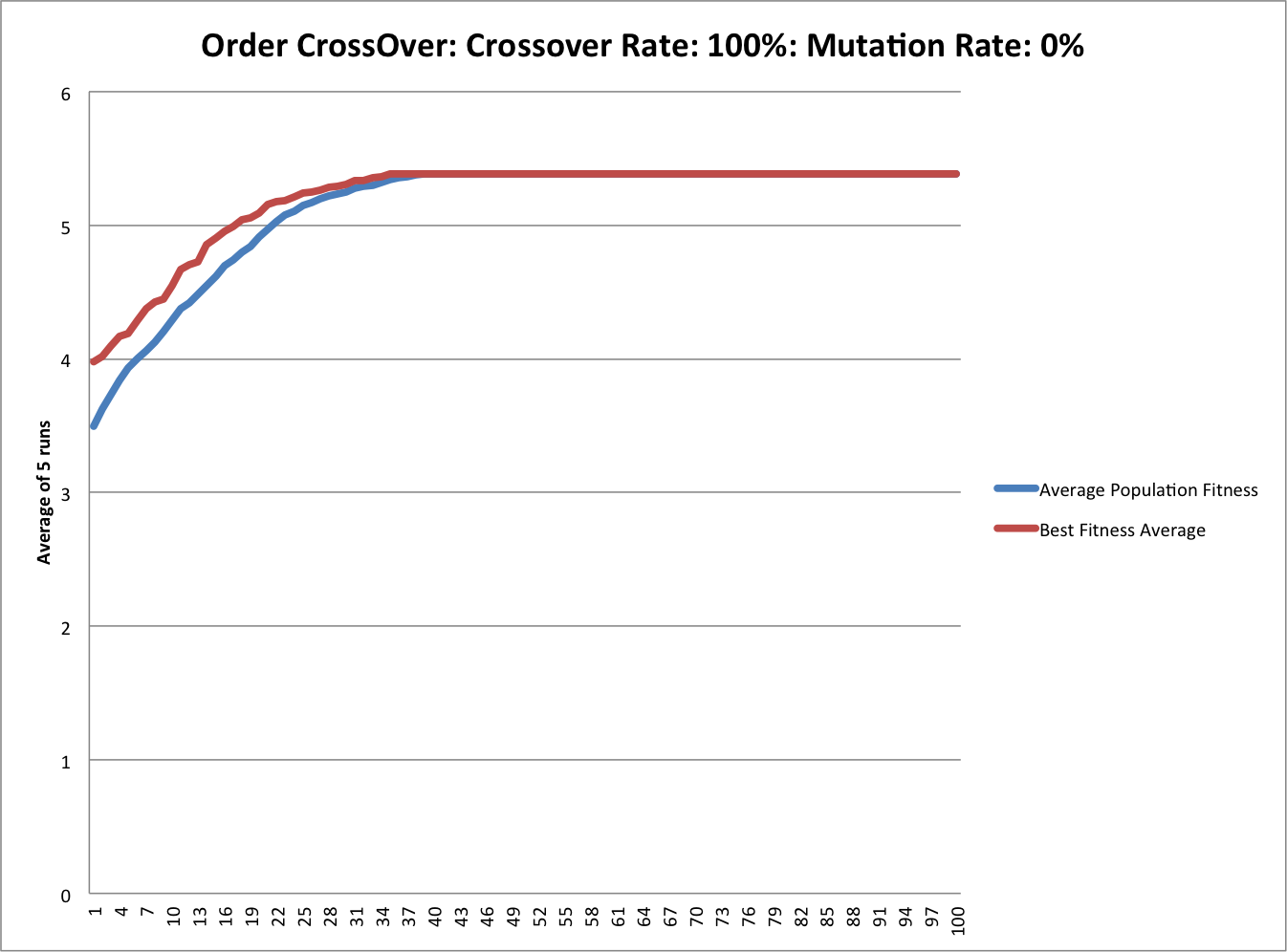
Run: 1 Seed: 0 AverageFitness: 5.121870660141537 Best Fitness: 9.053870529651427 Distance: 11045.0

Run: 2 Seed: 1 AverageFitness: 5.1574047062593165 Best Fitness: 9.350163627863488 Distance: 10695.0

Run: 3 Seed: 2 AverageFitness: 5.20806572162929 Best Fitness: 8.856611460455229 Distance: 11291.0

Run: 4 Seed: 3 AverageFitness: 5.42627994652102 Best Fitness: 9.176837661741764 Distance: 10897.0

Run: 5 Seed: 4 AverageFitness: 5.472342135500773 Best Fitness: 9.50660709192889 Distance: 10519.0



**Parameter: 5**

numberPerFun: 5

elitism: TRUE

elitismRate: 0.1

crossOver: 1.0

mutationRate: 0.0

generations: 10000

population: 50

UniformOrderCrossOver: FALSE

OrderCrossOver: TRUE

tournamentSize: 3

**Best of each run:**

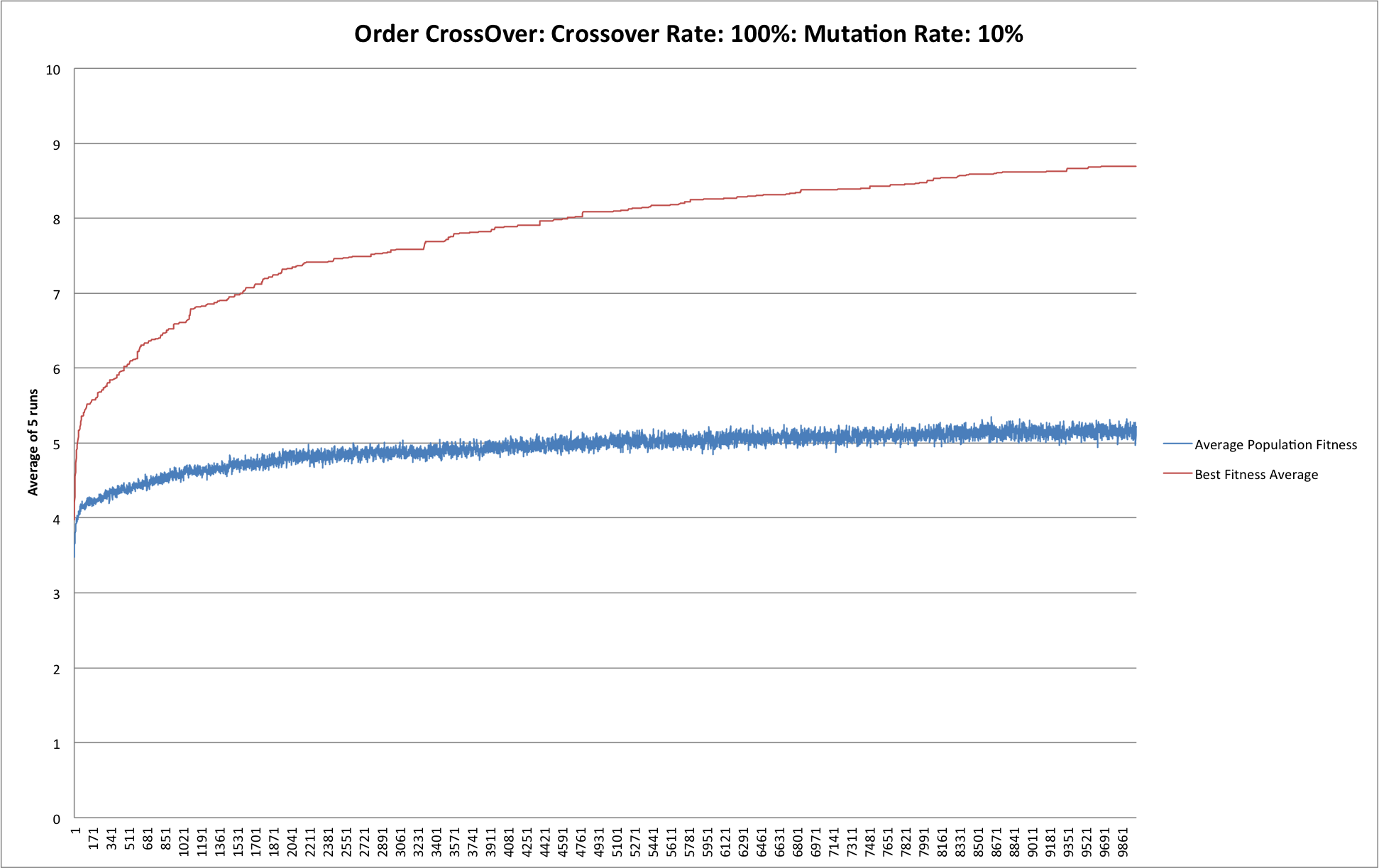
Run: 1 Seed: 0 AverageFitness: 5.417998591320363 Best Fitness: 5.417998591320366 Distance: 18457.0

Run: 2 Seed: 1 AverageFitness: 5.443954488540474 Best Fitness: 5.443954488540476 Distance: 18369.0

Run: 3 Seed: 2 AverageFitness: 4.974381933044816 Best Fitness: 4.974381933044819 Distance: 20103.0

Run: 4 Seed: 3 AverageFitness: 5.8885879166176025 Best Fitness: 5.888587916617595 Distance: 16982.0

Run: 5 Seed: 4 AverageFitness: 5.21349251863824 Best Fitness: 5.213492518638236 Distance: 19181.0



**Parameter: 6**

numberPerFun: 5

elitism: TRUE

elitismRate: 0.1

crossOver: 1.0

mutationRate: 0.1

generations: 10000

population: 50

UniformOrderCrossOver: FALSE

OrderCrossOver: TRUE

tournamentSize: 3

**Best of each run:**

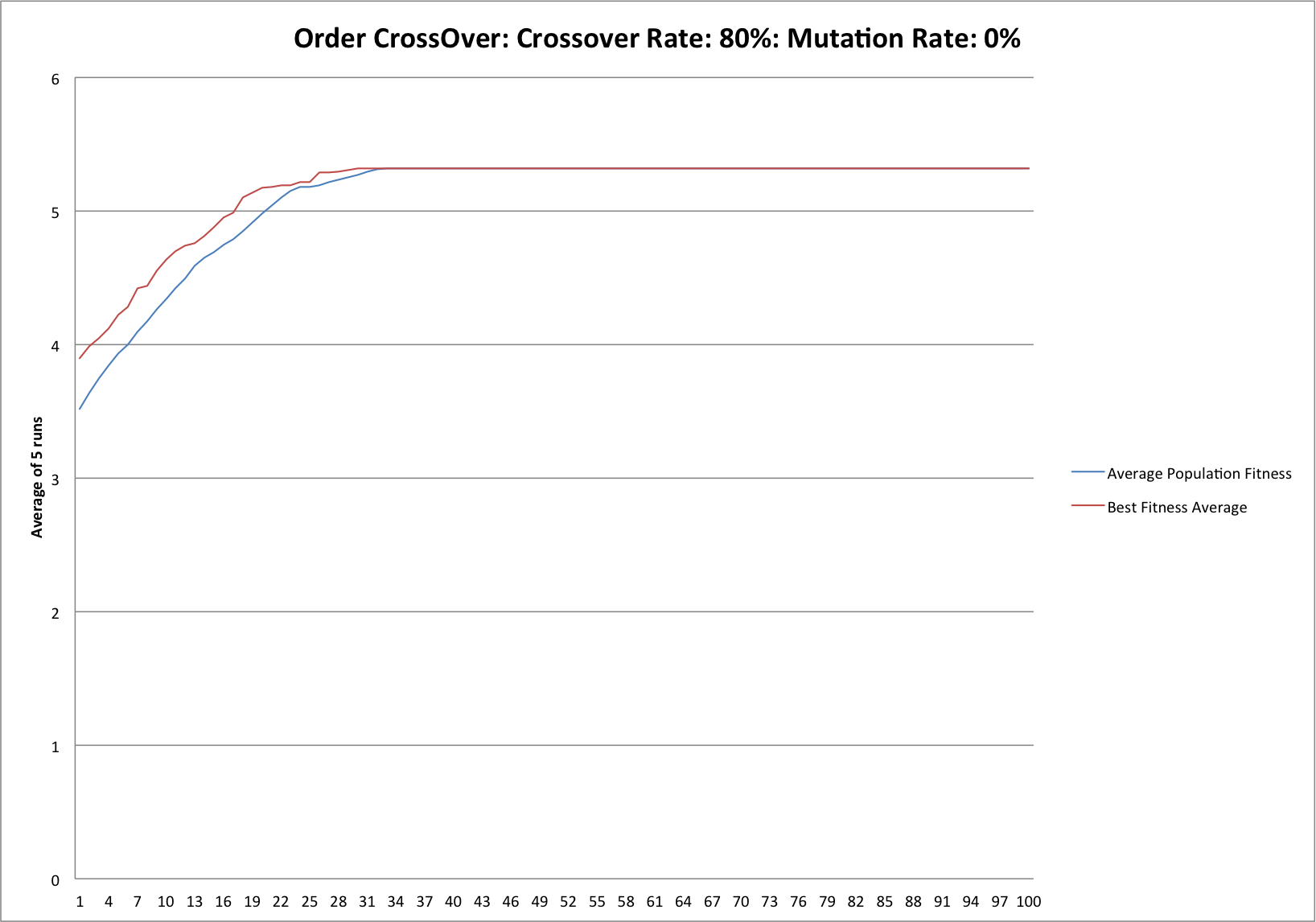
Run: 1 Seed: 0 AverageFitness: 5.3209199168595624 Best Fitness: 9.19793966151582 Distance: 10872.0

Run: 2 Seed: 1 AverageFitness: 5.253527930267097 Best Fitness: 8.56384345294168 Distance: 11677.0

Run: 3 Seed: 2 AverageFitness: 5.272809470254745 Best Fitness: 8.585901948999743 Distance: 11647.0

Run: 4 Seed: 3 AverageFitness: 5.12234056355969 Best Fitness: 8.284317786430288 Distance: 12071.0

Run: 5 Seed: 4 AverageFitness: 5.1217942063111215 Best Fitness: 8.820675663755845 Distance: 11337.0



**Parameter: 7**

numberPerFun: 5

elitism: TRUE

elitismRate: 0.1

crossOver: 0.8

mutationRate: 0.0

generations: 10000

population: 50

UniformOrderCrossOver: FALSE

OrderCrossOver: TRUE

tournamentSize: 3

**Best of each run:**

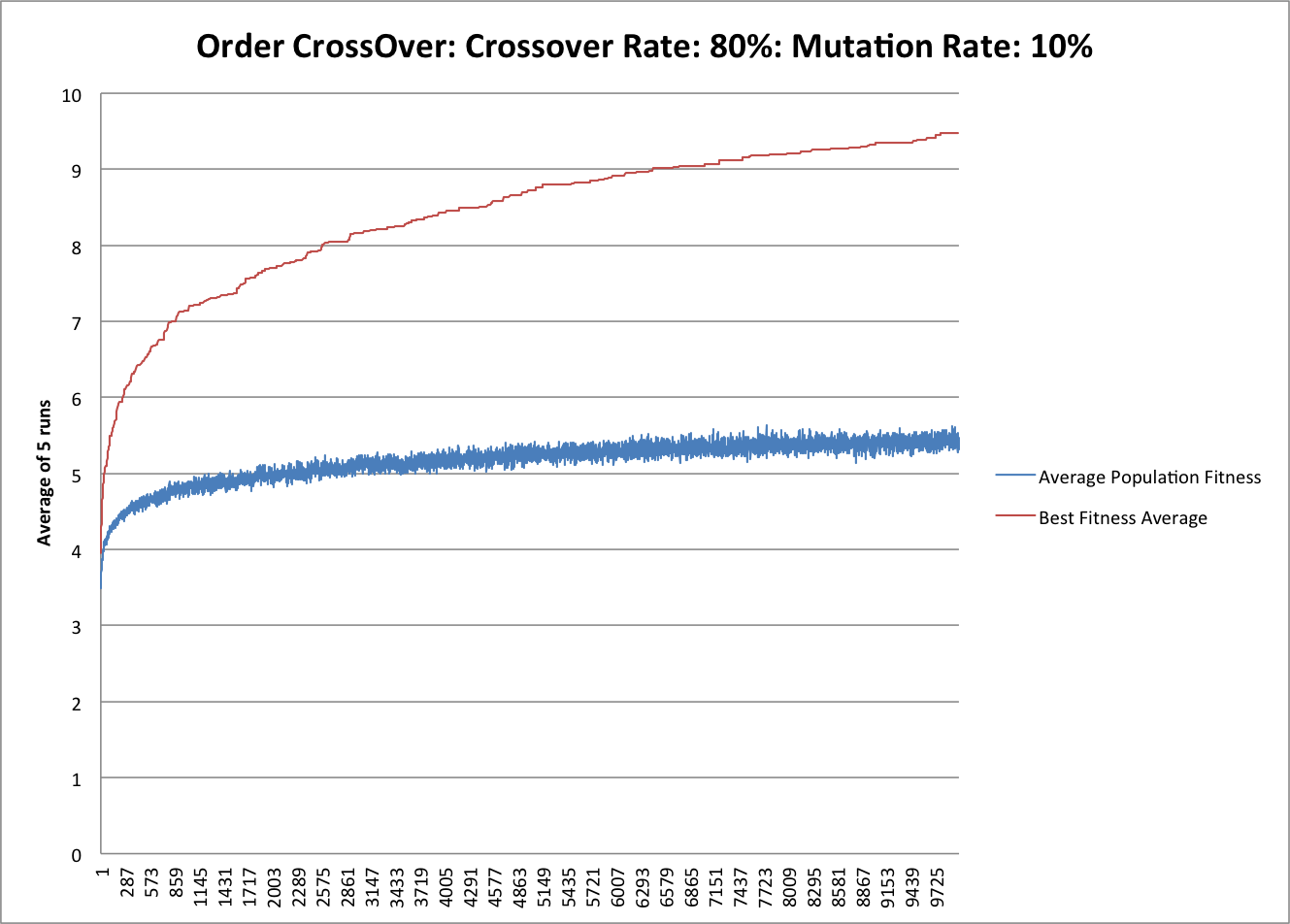
Run: 1 Seed: 0 AverageFitness: 5.527304886137523 Best Fitness: 5.52730488613752 Distance: 18092.0

Run: 2 Seed: 1 AverageFitness: 4.957612413861484 Best Fitness: 4.957612413861484 Distance: 20171.0

Run: 3 Seed: 2 AverageFitness: 5.234231876472132 Best Fitness: 5.234231876472128 Distance: 19105.0

Run: 4 Seed: 3 AverageFitness: 4.671150971599396 Best Fitness: 4.671150971599403 Distance: 21408.0

Run: 5 Seed: 4 AverageFitness: 6.211565935772403 Best Fitness: 6.211565935772408 Distance: 16099.0



**Parameter: 8**

numberPerFun: 5

elitism: TRUE

elitismRate: 0.1

crossOver: 0.8

mutationRate: 0.1

generations: 10000

population: 50

UniformOrderCrossOver: FALSE

OrderCrossOver: TRUE

tournamentSize: 3

**Best of each run:**

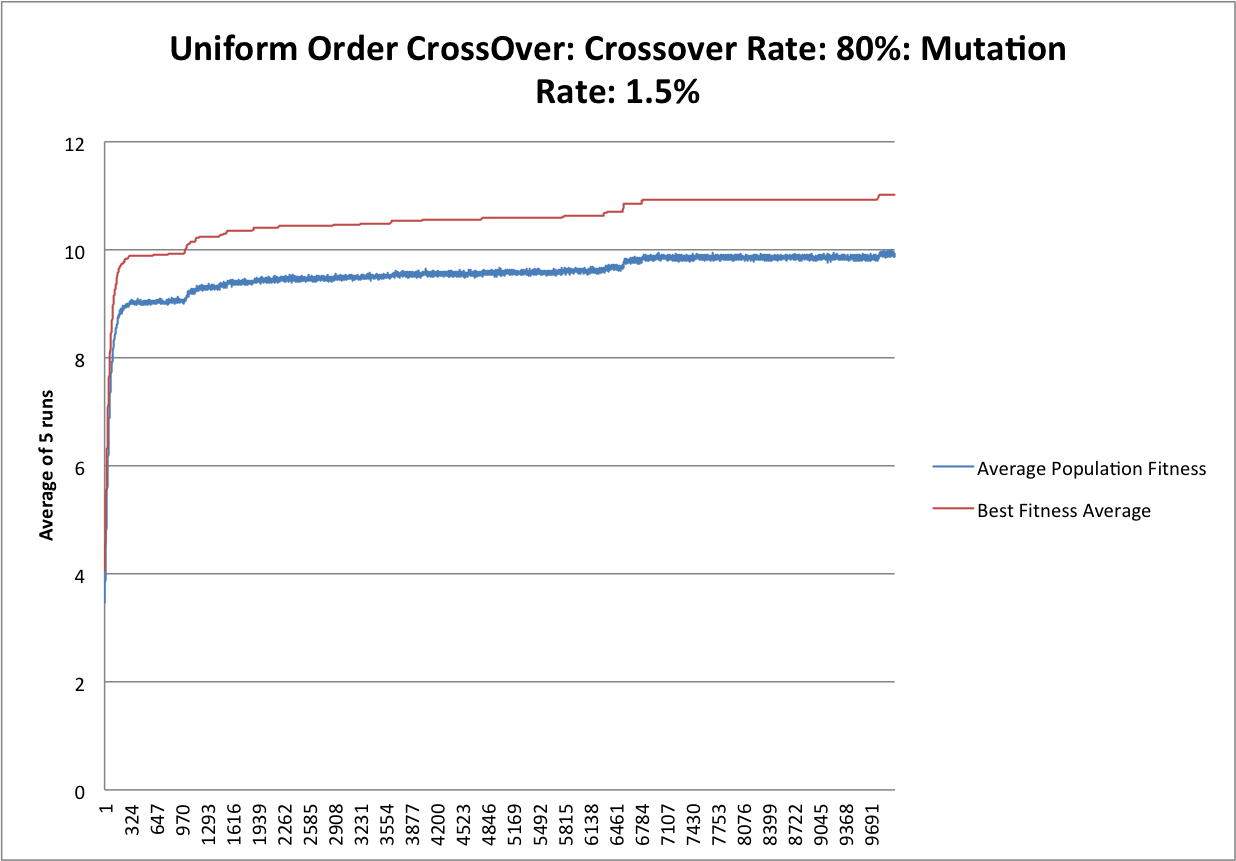
Run: 1 Seed: 0 AverageFitness: 5.810434886848925 Best Fitness: 10.488777008600797 Distance: 9534.0

Run: 2 Seed: 1 AverageFitness: 5.1278812734945936 Best Fitness: 9.17094644167278 Distance: 10904.0

Run: 3 Seed: 2 AverageFitness: 5.247913778685619 Best Fitness: 9.442870632672332 Distance: 10590.0

Run: 4 Seed: 3 AverageFitness: 5.170726307918058 Best Fitness: 8.968609865470851 Distance: 11150.0

Run: 5 Seed: 4 AverageFitness: 5.312791579774779 Best Fitness: 9.307520476545049 Distance: 10744.0



**Parameter: 9**

numberPerFun: 5

elitism: TRUE

elitismRate: 0.1

crossOver: 0.8

mutationRate: 0.015

generations: 10000

population: 500

UniformOrderCrossOver: TRUE

OrderCrossOver: FALSE

tournamentSize: 3

**Best of each run:**

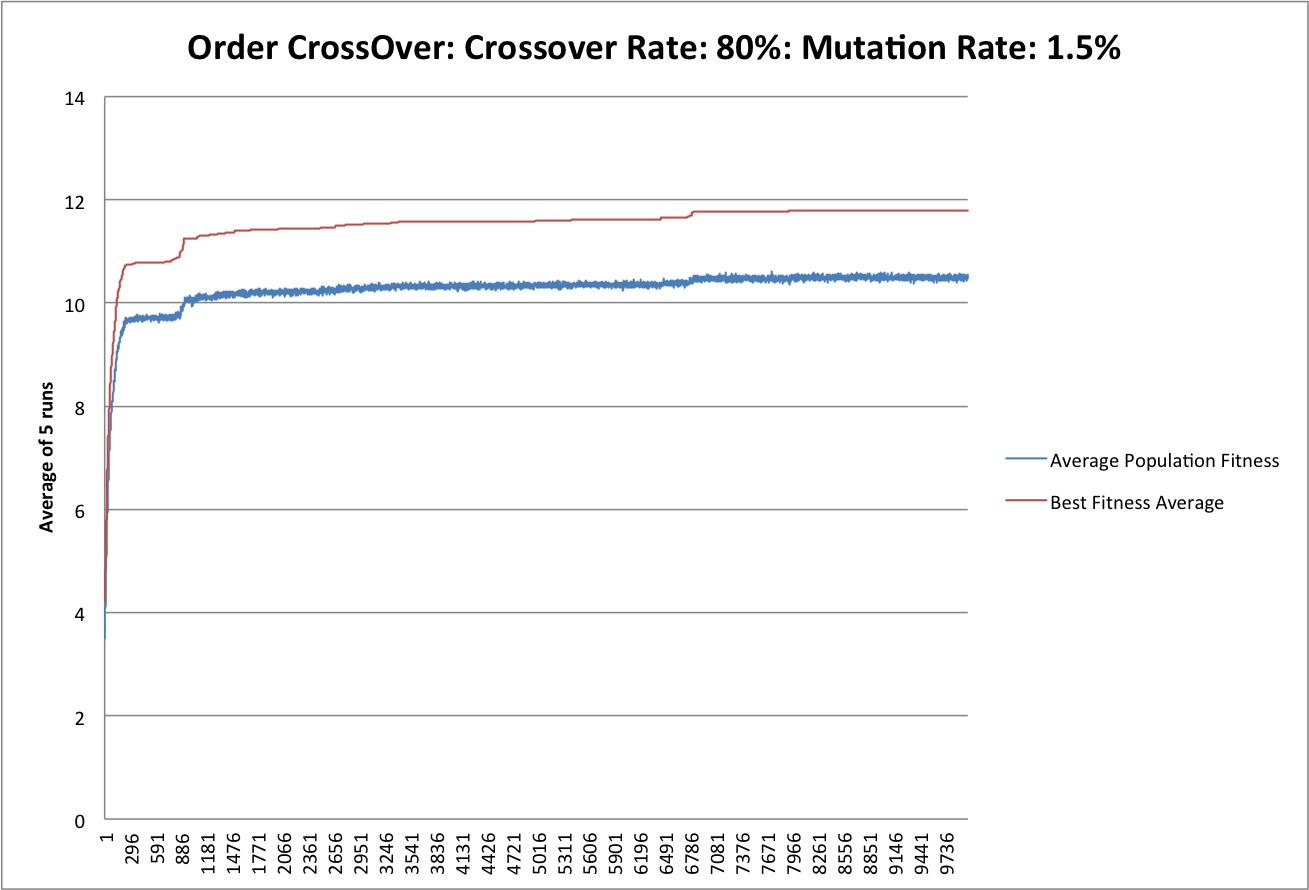
Run: 1 Seed: 0 AverageFitness: 10.002677973524229 Best Fitness: 11.138338159946537 Distance: 8978.0

Run: 2 Seed: 1 AverageFitness: 9.62191504017836 Best Fitness: 10.556317956296844 Distance: 9473.0

Run: 3 Seed: 2 AverageFitness: 9.49157788451153 Best Fitness: 10.520778537611783 Distance: 9505.0

Run: 4 Seed: 3 AverageFitness: 10.240770283871777 Best Fitness: 11.386927806877704 Distance: 8782.0

Run: 5 Seed: 4 AverageFitness: 10.189866456752462 Best Fitness: 11.453441759248655 Distance: 8731.0



**Parameter: 10**

numberPerFun: 5

elitism: TRUE

elitismRate: 0.1

crossOver: 0.8

mutationRate: 0.015

generations: 10000

population: 500

UniformOrderCrossOver: FALSE

OrderCrossOver: TRUE

tournamentSize: 3

**Best of each run:**

Run: 1 Seed: 0 AverageFitness: 11.107812464693144 Best Fitness: 12.47193813918683 Distance: 8018.0

Run: 2 Seed: 1 AverageFitness: 10.007502354321076 Best Fitness: 11.189437171310283 Distance: 8937.0

Run: 3 Seed: 2 AverageFitness: 10.371011801398376 Best Fitness: 11.617100371747211 Distance: 8608.0

Run: 4 Seed: 3 AverageFitness: 11.24428701579273 Best Fitness: 12.5203455615375 Distance: 7987.0

Run: 5 Seed: 4 AverageFitness: 9.920230812906471 Best Fitness: 11.148272017837236 Distance: 8970.0

**Conclusion**

While testing out both cross over methods **“Uniform Order Cross Over”** and **“Order Cross Over”:**

**100% Crossover, 0% Mutation:**

The average fitness in a given generation was always equal to the fittest individual.

**100% Crossover, 10% Mutation:**

Yielded a better result, progressed rapidly towards solution compared to the one with zero mutations.

**80% Crossover, 0% Mutation:**

The average fitness in a given generation was always equal to the fittest individual.

**80% Crossover, 10% Mutation:**

While it performed better than 0% Mutation instances, the results were still on average below or equal to 100% Crossover.

**My own parameter settings:**

While testing out my own parameters, further analysis on the algorithm, the observations made are:

Too high of a mutation rates give the fittest individual a chance to be mutated, causing the algorithm to potentially backtrack slightly.

Lower mutation rates lead to no differentiation between populations.

Higher crossover rates gives the populations more variance and improves the rate at which we progress towards the solution.

Lower crossover rates provide us with a more clustered set of results.

A more optimal solution is around 80% crossover and 1.5% mutation for both crossover methods.

**Bonus:**

I wanted to know if I could get better result if there are no changes for a few thousands generation for the best fitness so I wrote a method called “changeMatationRate()” to see if it will make it better. If there are no changes for a few 100 or 1000 generation, it will increase the mutation rate by 0.001 every time. After trying this out and testing it, it made very little different to the solution in the end. Because if it increases it too much, there’s a chance it will be worst off then before because it will be more random and make the better chromosome worse.