Title:

Genetic Algorithm as a Solution to the Travelling Salesman Problem and the Optimization of Mutation Rate in this Problem

Research Question:

What is the Impact of the Mutation Rate on the Effectiveness of the Genetic Algorithm in Travelling Salesman Problem?

Subject:

Computer Science

Word Count:

3548

Table of Contents:

Abstract

Although the structure of Genetic Algorithm (GA) are almost the same for solving different problems, their parameters vary a lot, including “population size”, “fitness function”, “type of crossover & mutation operators”, “crossover rate”, and “mutation rate”. Since the invention of the genetic algorithm, a lot of efforts have been put to understand how each parameter will affect the performance of the genetic algorithm, but no conclusive and definite conclusion has been drawn. In other words, the effect of these parameters on the performance is local to each specific problem.

Travelling Salesman Problem (TSP) is an NP-hard [1] problem in the field of combinatorial optimization. A genetic algorithm can be used to solve this problem. In this extended essay, the impact of different mutation rates on the genetic algorithm will be identified in order to find out the most preferable rate for Travelling Salesman Problem by conducting the controlled experiment.

Keywords: Genetic Algorithm (GA), Travelling Salesman Problem (TSP), Genetic Operators, Mutation Rate.

[1] https://en.wikipedia.org/wiki/NP-hard

1. Introduction

1.1 Travelling Salesman Problem (TSP)

1.1.1 Overview

TSP is a problem which asks to find the shortest possible route to visit all the cities on a plane exactly once and return to the starting city given the coordinates of all cities. A traditional approach, which is the permutation method, will not work for a large number of cities since the time complexity for TSP is $O(n!)$. For example, the total possible routes that can be taken for 60 cities approximately equal to the estimated number of atoms in the known, observable universe, which is far beyond the computation power of all existing computer. Therefore, researches have put tremendous efforts to come up with solutions to solve TSP for a large number of cities. Different approaches have been designed. Dynamic programming using graph theory can solve the problem and find the exact solution in time $O(n^2 2^n)$, but that is still a lot of time. Later, people have designed various heuristic and approximation algorithms, which can easily handle a large number of cities and give quick solutions, but the results given are usually suboptimal.

1.1.2 Applications

The TSP is not just a problem, it also has some applications in the real world. Naturally, it appears as a sub-problem in transportation and logistics, for example, designing a bus route for picking up students in a district, or a router for food delivery personnel. Slightly modified, TSP can be applicable in many other fields.

For example, the scheduling of a machine to drill holes in a circuit board or other areas. The TSP also appears in astronomy, as astronomers observing many sources will want to minimize the time spent moving the telescope between the sources. ("TSP Applications")

1.2 Genetic Algorithm (GA)

1.2.1 Background Information

Inspired by Charles Darwin's theory of natural selection, GA is proposed by Professor John Henry Holland in the 1970s in the University of Michigan. As a branch of the evolutionary algorithm (EA), GA is a metaheuristic (problem-independent heuristic) approach to search for high-quality solutions and optimize problems. Therefore, this algorithm is adaptive to a lot of problems across many fields of studies. For example, it has been applied to combinatorial optimization problems, machine learning, scheduling, Electronic circuit design, software engineering, economics, etc. ("Introduction To Genetic Algorithms")

1.2.2 Running Process

A genetic algorithm starts with a randomly generated *population* which represent a set of solutions to the target problem. A population consists of a certain number of *individuals*, each carrying encoded *genes* containing information of one solution to the target problem. After the first population is generated, the *fitness value* of each individual will be evaluated according to the *fitness function* defined by

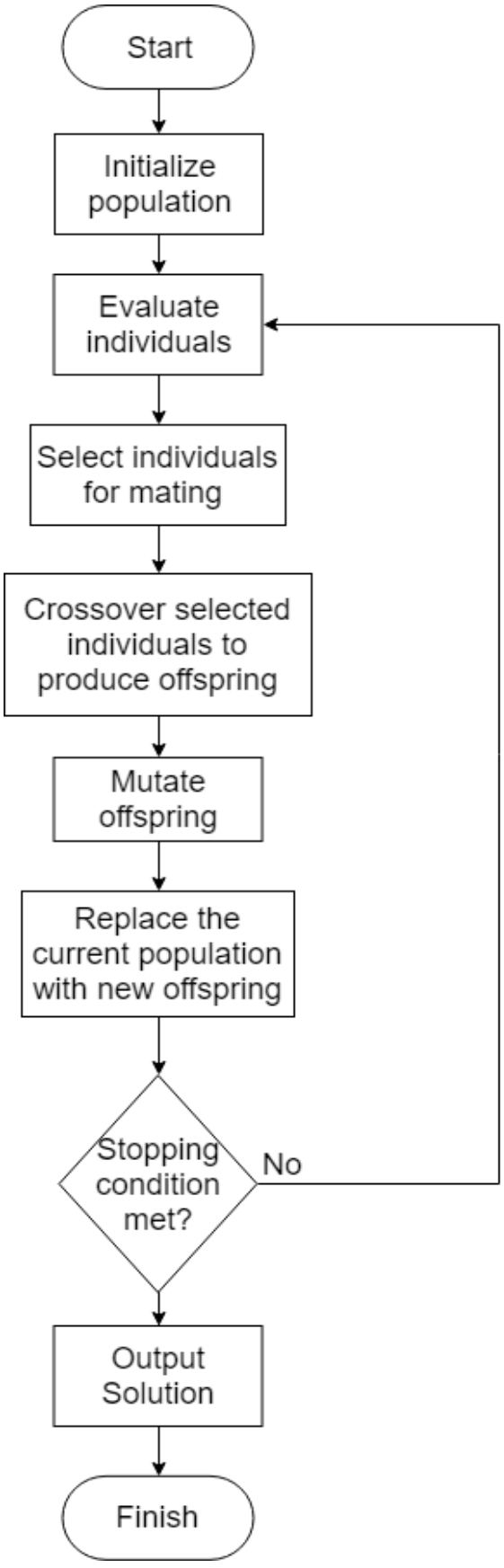


Figure 1. Flowchart of GA

programmer. Then, according to mechanism of natural selection, individuals with higher fitness value will be more likely to be selected to pass their genetic information to the next generation using genetic operators – *crossover* and *mutation*. In this way, a new generation with the exact same size as the first generation will be generated. Then, the individuals' fitness value in the new generation will be evaluated. Again, a new generation will be selected. While the process keeps repeating, the individuals in each population will gradually evolve and obtain higher fitness values until it reaches the stopping condition set by the programmer. Then, the genes of the individual with the highest fitness value in the last generation will be decoded to represent the final solution of the problem.

1.2.3 Implication of Increasing and Decreasing the Mutation Rate

Mutation adds variety to the population and prevents it from being trapped in a local maximum. Also, it enriches the population by mutating the incomplete genetic information in the initial population. However, to achieve its maximum potential, a proper mutation rate needs to be selected. A mutation rate that is too low would not achieve its functionalities such as diversify the population and escape the local optimum; a mutation rate that is too high would disrupt the population, and the excessive randomness would cause reverse evolution in individuals with high fitness values. Hence, it's very important to find a proper value for mutation rate which can achieve the highest efficiency.

1.2.4 Advantages of GA

Comparing with other traditional optimizations algorithms, such as *random walk, exhaustive search,* and *steepest descent*, GA has certain advantages:

1. GA starts with a set of solutions instead of one, which gives it a higher chance of finding the global optimum, while traditional optimization algorithms start with one solution, which can fall into local optimum rather easily.
2. GA uses fitness value as the only information for searching. It does not need derivatives or other tools, which reduces the complexity.
3. GA starts with random solutions and uses probability to guide itself to find the optimum.

2. Genetic Operators

2.1 Selection Operator

Selection is a process in a genetic algorithm during which individuals in a population get selected by certain rules for generating the next generation. Some common rules are:

- **Roulette Wheel Selection**

  Roulette Wheel Selection is also called fitness proportionate selection, which means that the probability that an individual being selected is proportionate to the fitness value of that individual. For any individual i, its probability of being selected can be denoted as

$$\mathrm{P}(i\ is\ selected) = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

  where f_j is the fitness value of individual j, N is the total number of individuals in the population.

  It is called Roulette Wheel Selection because this can be visualized as Roulette Wheel. In Figure 2, each letter represents an individual and its area represent the ratio of its fitness value to the sum of all fitness values.

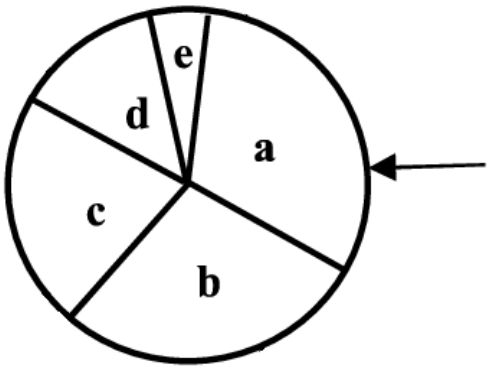


Figure 2. Visualization of Roulette Wheel Selection

- **Stochastic Universal Sampling**

  Stochastic Universal Sampling is similar to Roulette Wheel Selection, but it displays zero bias and evenly spread intervals. It can also be visually represented by a roulette wheel, however, with N pointers, where N is the number of individuals to be selected in each operation.

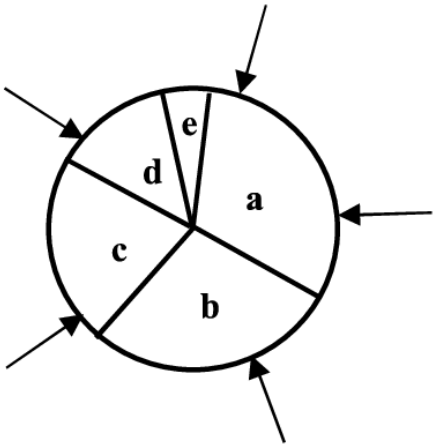


Figure 3. Visualization of Stochastic Universal Sampling

In Figure 3, it shows a Stochastic Universal Sampling which selects 5 individuals at once.

- **Tournament Selection**

  Tournament Selection is like running multiple tournaments. Each time, a certain number – decided by the programmer – of individuals are randomly chosen from the population. And the winner (the individual with the highest fitness value in the tournament) will be selected for mating. This process will keep repeating until enough individuals are selected. The process is demonstrated in the diagram below.

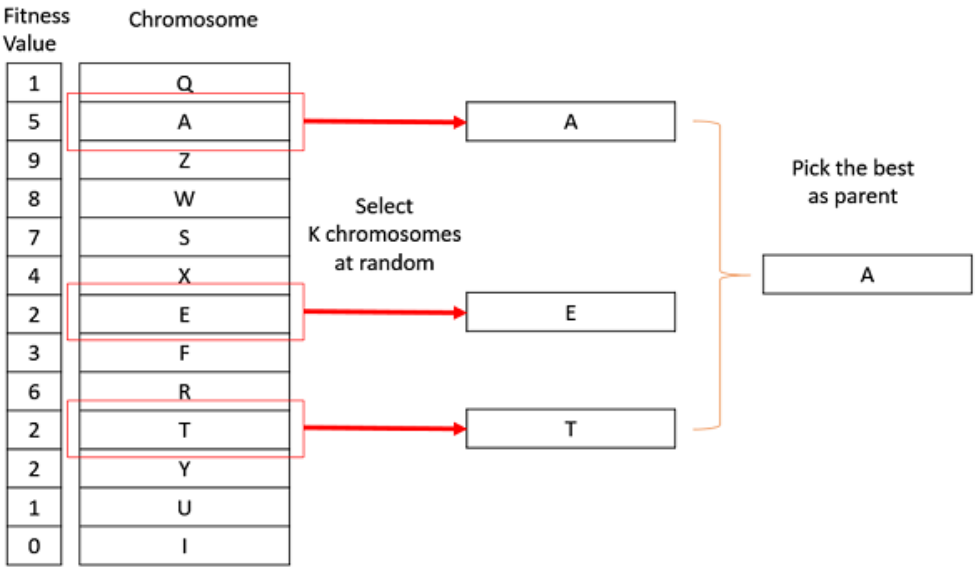


Figure 4. Visualization of Tournament Selection

- **Elitism**

  Elitism is an additional selection operator that passes down the individual with the highest fitness value directly to the next generation to prevent the loss of best individuals in the selection process.

2.2 Crossover Operator

Crossover is a process in genetic algorithm during which selected individuals pair up and produce new offspring by combining their genetic information according to the operator chosen by the programmer. Some common crossover operators are:

- **Single-point crossover**

  Single-point crossover is the most intuitive and commonly used technique for crossover operation. It is easy to use and fast. One point in the genes is randomly selected first, then the section from the selected point to the end of the genes of two parents will be switched. For example, consider two genes

encode with integer strings with the randomly selected point between the third and the fourth digit.

Parent 1: 1 2 3 | 4 5 6 7 8
Parent 2: 8 2 4 | 7 5 1 3 6

According to the rule, after switching the later section, we have two new genes.

Child 1: 1 2 3 | 7 5 1 3 6
Child 2: 8 2 4 | 4 5 6 7 8

- **K-point crossover**

  K-point crossover is somewhat similar to the single point crossover, except now we have k breaking points. For example, if k = 3, and the parents are:

Parent 1: 1 2 3 | 4 5 | 6 7 | 8
Parent 2: 8 2 4 | 7 1 | 5 3 | 6

  For all sections with an odd number count, their genes will remain the same, for all sections with an even number count, their genes will be switched to produce new genes. The genes of two children will be:

Child 1: 1 2 3 | 7 1 | 6 7 | 6
Child 2: 8 2 4 | 4 5 | 5 3 | 8

However, in some problems such as TSP, the genes carried by individual will not always be valid. It will only be valid if no gene – In the case of TSP, city – will be repeated in each individual solution. This is called *Permutation Encoding*. In this case, the crossover operators mentioned above will not be applicable. Some commonly used crossover operators especially designed for permutation encoding are:

- **Partially Mapped Crossover (PMX)**

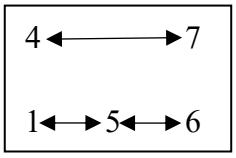
  Partially Mapped Crossover, proposed by Goldberg[2], is one of the most widely used crossover operators for GA with permutation encoding. First, two random points are selected for both parents.

  Parent 1: 1 2 3 | 4 5 6 | 7 8
  Parent 2: 8 2 4 | 7 1 5 | 3 6

  Then, the section in between for parent 1 and 2 will be switched,

  Parent 1: 1 2 3 | 7 1 5 | 7 8
  Parent 2: 8 2 4 | 4 5 6 | 3 6

  and a map is created between Parent 1 and Parent 2 in the middle section.



  To legalize two parents, the middle section will remain unchanged, and all the digits in the left and right section of Parent 1 and Parent 2 of 1, 4, 5, 6, 7 will be mapped according to the diagram above. We get,

  Child 1: 6 2 3 | 7 1 5 | 4 8
  Child 2: 8 2 7 | 4 5 6 | 3 1

  In this way, we get two children after crossover operation without repeated genes.

- **Cycle Crossover (CX)**

  The Cycle Crossover operator, identifies a number of so-called cycles between two parents. ("Cycle Crossover Operator Tutorial")

  The cycle crossover follows the following procedure:

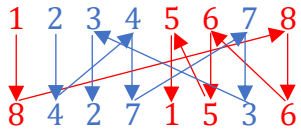
[2] https://en.wikipedia.org/wiki/David_E._Goldberg

1. Find the first unused position in Parent 1, and find its allele (the corresponding position in Parent 2).
2. Find the position in Parent 1 that contains the allele found in Step 1.
3. Add this allele to this cycle.
4. Repeat Step 1~3 until the starting point is reached.
5. Repeat Step 1~4 until all cycles are done.

For example, if we have parents:

Parent 1: 1 2 3 4 5 6 7 8

Parent 2: 8 4 2 7 1 5 3 6



In this example, the red line represents the first cycle, and the blue line represents the second cycle. Then, to form the child, combine the red part in Parent 1 and the blue part in Parent 2.

Child: 1 4 2 7 5 6 3 8

- **Ordered Crossover (OX)**

Ordered Crossover, proposed by Davis in 1989. Similar to PMX, it also starts with selecting two random breaking points in the genes. But instead of mapping genes to other genes, it fills the blank one by one from the other parent.

Using the same example from PMX,

Parent 1: 1 2 3 | 4 5 6 | 7 8
Parent 2: 8 2 4 | 7 1 5 | 3 6

First, we take only the middle section of two parents,

Parent 1: X X X | 4 5 6 | X X
Parent 2: X X X | 7 1 5 | X X

Then, starting from the position after the second breaking point, we fill Parent

1 with genes Parent 2, also starting from the position after the second break point. If encountered a gene in Parent 2 which already exists in Parent 1, we will skip it. Then, repeat the process from Parent 2, we can get two children:

Child 1: 2 7 1 | 4 5 6 | 3 8
Child 2: 3 4 6 | 7 1 5 | 8 2

2.3 Mutation Operator

Mutation is a process in genetic algorithm during which the genes of newly generated individuals in the crossover process have the probability to be altered. that immediately follows the crossover process. The mutation process is important because it adds variety to the population and prevents it from being trapped in a local maximum. Also, it enriches the population by mutating the incomplete genetic information in the initial population. Some commonly used mutation operators are:

- **Swap Mutation**

  This is the most common and simple mutation method. Two random positions are chosen in the genes and the elements at these two positions are swapped. For example,

  Before Mutation: 1 2 3 4 5 6 7 8
  After Mutation: 1 6 3 4 5 2 7 8

- **Insert Mutation**

  During insert mutation, two random positions are chosen in the genes, and the element in the second position will be moved next to the element in the first

position. This mutation operator will preserve most of the order and the adjacency information. For example,

Before Mutation: 1 2 3 4 5 6 7 8
After Mutation: 1 2 6 3 4 5 7 8

- **Inversion Mutation**

During inversion mutation, two random positions are chosen in the genes, and the substring between the two positions will be inverted. This mutation operator will preserve most of the adjacency information, but this will disrupt the order in the original genes. For example,

Before Mutation: 1 2 3 4 5 6 7 8
After Mutation: 1 6 5 4 3 2 7 8

3. Implementation of Genetic Algorithm

3.1 Gene Encoding

Firstly, a *City* Class is created. It contains two float type properties – x, y – indicating its coordinate on the canvas. The following diagram is the UML class diagram of the *City* Class.

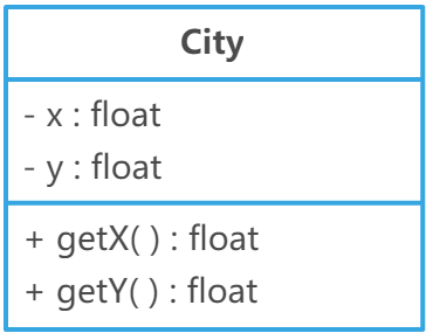


Figure 5. UML Diagram of the *City* Class

Therefore, each City object represents a gene, and the individual is made up from a list of genes. In this case, each individual will be encoded by a list of City objects, denoting the sequence of cities that the travelling salesman travels.

3.2 Fitness Function

The fitness function is based on the total distance travelled. The shorter the distance is, the better the solution is, so the higher the fitness value should be. Therefore, a fitness value should have an inverse relationship with the distance travelled. After multiple trials, I have designed the fitness function as

$$\boldsymbol{f(d) = \left(\frac{10000}{d}\right)^4}$$

where d is the total distance travelled. The factor 10000 is to get rid of the decimal places in the fitness value so that it is easier to observe the performance, and the 4th power is to bring proportionately larger change to the fitness value when there is a change in distance travelled. Therefore, it increases the probability that individuals with a shorter distance to be selected in the selection process.

3.3 Parameters Selected

For this experiment, I have chosen the **population size as 1000**, so that the initial population will be large enough to have enough diversity for later generations. I have selected **0.9 as my crossover rate** because it is generally acknowledged that the crossover rate should be relatively high, and 0.9 will generally give good results.

3.4 Operators Selected

Selection Operator: Roulette Wheel Selection with Elitism

Crossover Operator: Order Crossover (OX)

Mutation Operator: Inversion Mutation

I have chosen Roulette Wheel Selection with Elitism as my selection operator because it is proven to be applicable and efficient in most cases. For crossover operation, I have selected Order Crossover because it is proven to be one of the best operators that produce good results in TSP.[3] For mutation operator, I have chosen Inversion Mutation because the experiment conducted by Otman, Jaafar, and Chakir has shown that Inversion Mutation has the best performance among all other tested operators.[4]

4. Experiment

4.1 Test Environment

Operating System: Windows 10 Home 64 bits

Software Platform: Processing 3.4[5]

Hardware Platform: Dell XPS 13 (9360)

- CPU: Intel(R) Core(TM) i5-8250U
- RAM: 8.00 GB
- Display Adapter: Inter(R) UHD Graphics 620

TSP Test Case: Berlin52[6]

[3] Abdoun, Otman, and Jaafar Abouchabaka. "A Comparative Study Of Adaptive Crossover Operators For Genetic Algorithms To Resolve The Traveling Salesman Problem". International Journal Of Computer Applications, vol 31, no. 11, 2011, p. 56.
[4] Abdoun, Otman et al. "Analyzing the Performance of Mutation Operators to Solve the Travelling Salesman Problem". LaRIT Laboratory.
[5] https://processing.org/
[6] https://github.com/pdrozdowski/TSPLib.Net/blob/master/TSPLIB95/tsp/berlin52.tsp

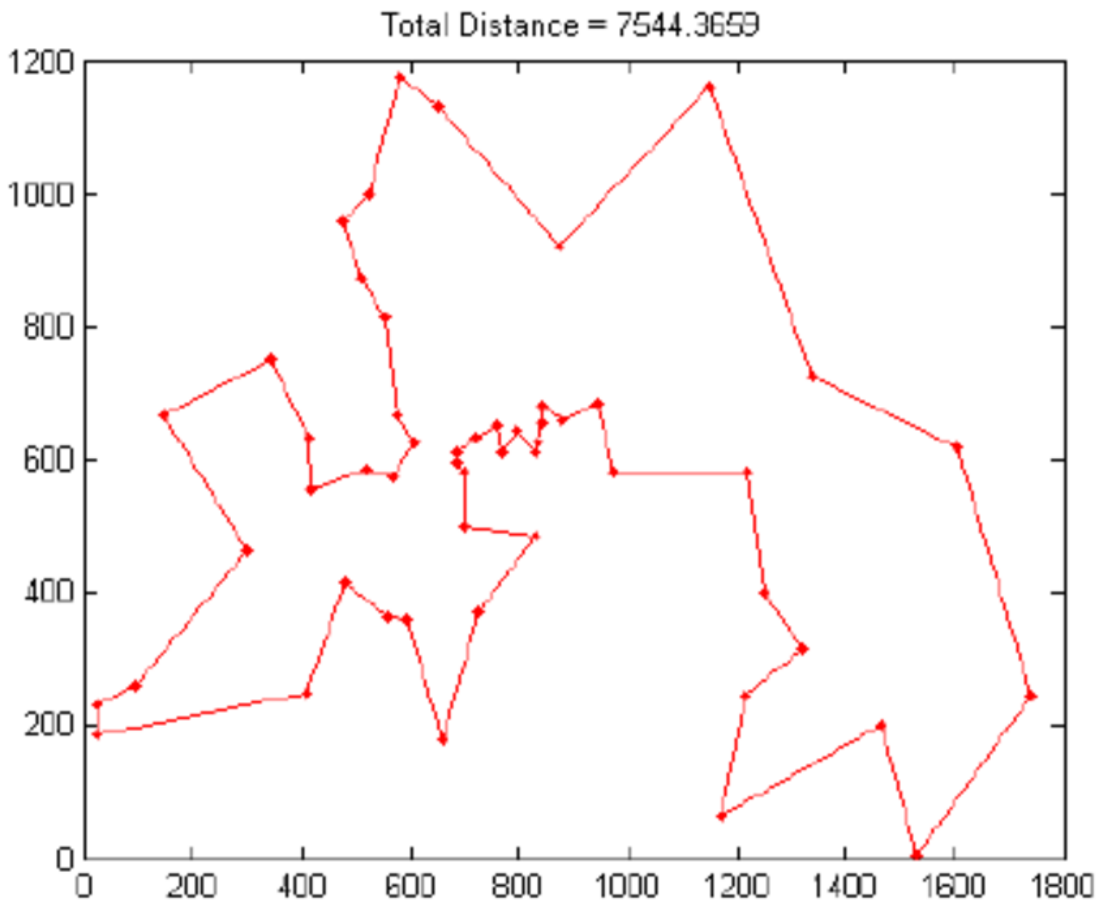


Figure 6. Optimum Solution for Berlin52 Test Case (Abdoun)

4.2 Test Method (For testing code, refer to Appendix A)

I have selected 5 mutation rates to be tested: 10%, 30%, 50%, 70%, 90%. The reason for this selection is that it covers the entire range of the probability, so that the best mutation rate obtained will be more reliable.

The stopping condition was set to be 5000 generations because after examining each mutation rate, the shortest distance obtained always become very stable and does not change after 5000 generations.

Because of the random nature of the genetic algorithm, 10 trials were run for each mutation rate. For each trial, the shortest distance of each 100 generations was

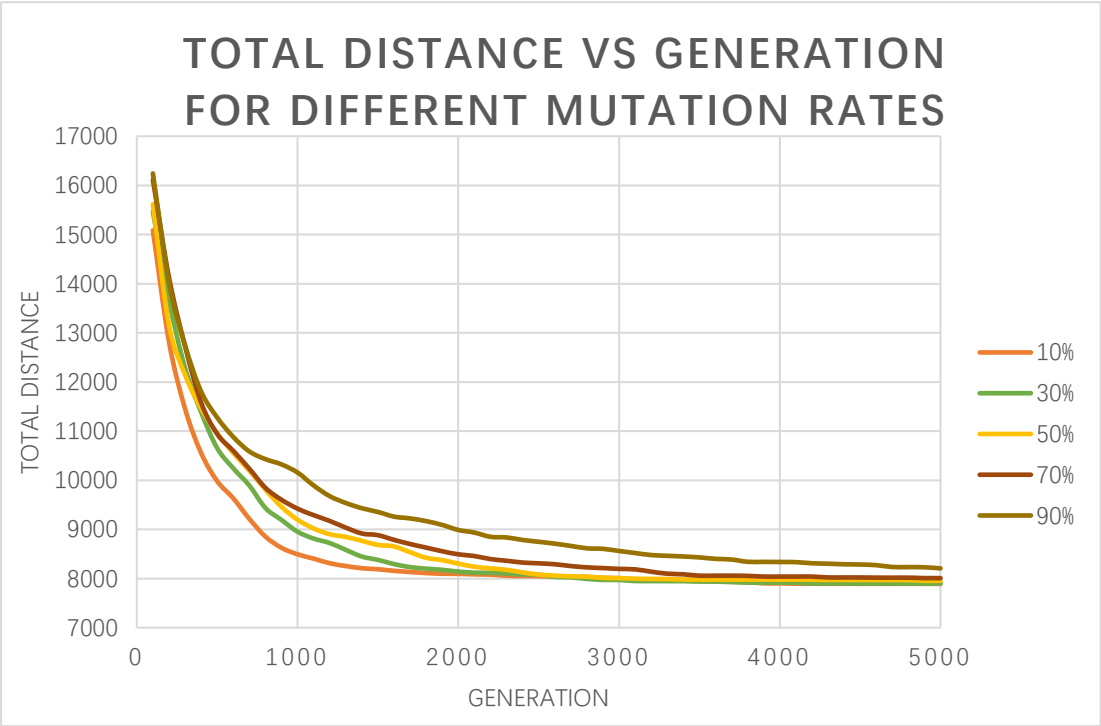
collected and saved in a text file. For each mutation rate, the average distances of 10 trials for every 100 generations were calculated to plot the graph, and the number of optimum solutions obtained in 10 trails was displayed.

4.3 Test Result (For complete test data, refer to Appendix B)

Table 1. Average Minimum Distance and Number of Optimum Solutions for Different Mutation Rates

Mutation Rate	Avg. Minimum Distance	Number of Optimum Solutions
10%	7893.38	3
30%	7896.47	1
50%	7955.27	0
70%	8009.84	0
90%	8207.79	0

Figure 7. Total Distance vs. Generation for Different Mutation Rates



4.4 Discussion

Using data obtained in the testing process, the table and diagram above are drawn. The table shows how good each mutation rate is in terms of getting better solutions. As is shown in Table 1, there is an inverse correlation between the mutation rates and their ability to obtain better solutions. As mutation rate increases, average shortest distance gradually increases from 7893.38 with 10% mutation rate to 8009.84 with 70% mutation rate, followed by a sharp increase to 8207.79 with 90% mutation rate. Therefore, it demonstrates that the effect of the mutation rate for getting better solutions is limited within certain ranges – in this case 10% to 70%. As for the number of optimum solutions, GAs with the lower mutation rate have performed much better as 3 out 10 times found in 10% mutation, 1 out of 10 times found in 30% mutation rate, while 0 time found in 50%, 70%, 90% mutation rates.

The graph is a scatter plot with smooth curves. Each curve represents a mutation rate with 50 data points consisting of the average shortest distances in every 100 generations in 10 trials. The graph demonstrates how efficient each mutation rate is in terms of reducing the distance. According to the graph, the curve with the lower mutation rate descends more quickly with a sharper slope in the beginning, and a flatter slope in the end. Although most of the curves (except for 90% mutation rate) descend to the approximately same level, it takes longer to reduce the distance to the desirable level as the mutation rate increases. Therefore, the one with 10% mutation rates is more capable of finding better solutions effectively.

5. Conclusion

In this paper, we have compared 5 different mutation rates – 10%, 30%, 50%, 70%, 90%. According to the experiment result, 10% turned out to be the best mutation rate for this TSP problem, as it resulted in the best performance both in terms of obtaining the optimum solution and effectively reducing the distance. Also, an inverse correlation between the mutation rates and their performance has been demonstrated. However, this experiment is subject to certain limitations. The experiment only tested 5 mutation rates ranging from 10% to 90%. Therefore, it is unable to pinpoint the specific value of the best mutation rate. Moreover, the result obtained is also subject to the changes in test environment and different combination of the operators used. Hence, the result is not applicable in a broad sense. More future studies need to be done to be able to obtain a formula to find the best mutation rate in a universal setting.

Works Cited

"TSP Applications". Math.Uwaterloo.Ca, 2019, http://www.math.uwaterloo.ca/tsp/apps/index.html.

"Introduction To Genetic Algorithms". Doc.Ic.Ac.Uk, 2019, https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol1/hmw/article1.html.

"Tournament Selection". 2018, https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm.

"Cycle Crossover Operator Tutorial". Rubicite.Com, 2018, http://www.rubicite.com/Tutorials/GeneticAlgorithms/CrossoverOperators/CycleCrossoverOperator.aspx.

Abdoun, Otman. The Optimum Solution of Berlin52. 2018, https://www.researchgate.net/figure/The-optimal-solution-of-Berlin52_fig2_221901574.

Tournament Selection. 2019, https://www.tutorialspoint.com/genetic_algorithms/images/tournament_selection.jpg.

Appendix A: Testing Code (Processing 3.4)

```
static int cityNum = 52;
City[] cities = new City[cityNum];
Population population;
BufferedReader reader;
String[] line;
int rateCount = 9;
int trialCount = 9;
String fileName = "0." + rateCount + "_" + (trialCount) + ".txt";
PrintWriter pw = createWriter("C:/Users/yuhao/Desktop/TSP/TSP_GeneticAlgorithm/" +
fileName);

void setup() {
  size(1800, 1200);
  frameRate(1000);
  reader = createReader("Berlin52.txt");
  try {
    for (int i = 0; i < cityNum; i++) {
      line = reader.readLine().split(",");
      float x = Float.valueOf(line[0]);
      float y = Float.valueOf(line[1]);
      City city = new City(x, y);
      this.cities[i] = city;
    }
  } catch (Exception e) {
    e.printStackTrace();
  }
  population = new Population(cities, 999, 0.9, (float)rateCount/10);
  population.calcFitness();
}

void draw() {
  background(0);
  fill(255,150,0);
  for (City city : this.cities) {
    ellipse(city.getX(), city.getY(), 10, 10);
  }
  stroke(255,150,0);
  strokeWeight(3);
  noFill();
  beginShape();
  City[] bestRoute = population.getBestRoute();
```

```
  for (City city : bestRoute) {
    vertex(city.getX(), city.getY());
  }
  vertex(bestRoute[0].getX(), bestRoute[0].getY());
  endShape();

  population.naturalSelection();
  population.calcFitness();
  if (population.getGenerationCount() % 100 == 0) {
    pw.println(population.getShortestDistance());
  }
  if (population.getGenerationCount() == 5000) {
    pw.flush();
    pw.close();
    trialCount++;
    if (trialCount == 11) {
      rateCount += 2;
      trialCount = 1;
      if (rateCount == 11) {
        System.exit(0);
      }
    }
    fileName = "0." + rateCount + "_" + (trialCount) + ".txt";
    pw = createWriter("C:/Users/yuhao/Desktop/TSP/TSP_GeneticAlgorithm/" + fileName);
    population = new Population(cities, 999, 0.9, (float)rateCount/10);
    population.calcFitness();
  }
}

public class City {
  private float x;
  private float y;
  City(float x, float y) {
    this.x = x;
    this.y = y;
  }
  public float getX() {
    return x;
  }
  public float getY() {
    return y;
  }
}
```

```
public class DNA {
  private City[] genes;
  private double fitness;
  private double distance;

  private double getFitness() {return this.fitness;}
  private double getDistance() {return this.distance;}

  private City[] getGenes() {
    return genes;
  }

  public DNA(City[] cities, boolean firstGeneration) {
    this.genes = deepCopy(cities);
    if (firstGeneration) {
      shuffleGenes(genes.length);
    }
  }

  private void shuffleGenes(int n) {
    for (int i = n-1; i > 0; i--) {
      int j = floor(random(i));
      swap(genes, i, j);
    }
  }

  private double calcFitness() {
    double distance = dist(genes[0].getX(), genes[0].getY(), genes[genes.length-1].getX(),
genes[genes.length-1].getY());
    for (int i = 1; i < genes.length; i++) {
      distance += dist(genes[i-1].getX(), genes[i-1].getY(), genes[i].getX(), genes[i].getY());
    }
    this.distance = distance;
    this.fitness = pow((float)(10000 / distance), 4);
    return this.fitness;
  }

  private DNA crossover(DNA parent2, int start, int end, float crossoverRate) {
    if (random(1) < crossoverRate) {
      final int count = end - start + 1;
      City[] child = new City[cityNum];
      arrayCopy(subset(this.genes, start, count), 0, child, start, count);
      int insertIndex = end + 1;
      for (int k = 0; k < cityNum; k++) {
```

```
            int index = (k + end + 1) % cityNum;
            City city = parent2.getGenes()[index];
            if (contains(child, city)) {
               continue;
            } else {
               child[insertIndex % cityNum] = city;
               insertIndex++;
            }
         }
         return new DNA(child, false);
      } else {
         return new DNA(this.genes, false);
      }
   }

   private DNA mutate(float mutationRate) {
      City[] newGenes;
      if (random(1) < mutationRate) {
         int i = floor(random(genes.length));
         int j = floor(random(genes.length));
         int start = min(i,j);
         int end = max(i,j);
         int count = end - start + 1;
         City[] before = (City[]) subset(genes,0,start);
         City[] after = (City[]) subset(genes, (end+1));
         City[] selectedSection = (City[]) subset(genes, start, count);
         selectedSection = (City[]) reverse(selectedSection);
         newGenes = (City[]) concat(before, selectedSection);
         newGenes = (City[]) concat(newGenes, after);
      } else {
         newGenes = this.genes;
      }
      return new DNA(newGenes, false);
   }

   void swap(City[] cities, int i, int j) {
      City temp = cities[i];
      cities[i] = cities[j];
      cities[j] = temp;
   }

   public City[] deepCopy(City[] genes) {
      City[] newGenes = new City[TSP_GeneticAlgorithm.cityNum];
      for (int i = 0; i < genes.length; i++) {
```

```
      newGenes[i] = genes[i];
    }
    return newGenes;
  }

  public boolean contains(City[] genes, City city) {
    boolean contains = false;
    for (City gene: genes) {
      if (city.equals(gene)) {
        contains = true;
        break;
      }
    }
    return contains;
  }
}

public class Population {
  private City[] cities;
  private int populationNum;
  private float mutationRate;
  private float crossoverRate;
  private DNA[] population = new DNA[populationNum];
  private DNA bestOne;
  private double shortestDistance;
  private double bestFitness;
  private double sumFitness;
  private int generationCount = 1;

  City[] getBestRoute() {
    return bestOne.getGenes();
  }

  public double getShortestDistance() {
    return this.shortestDistance;
  }
  public int getGenerationCount() {
    return this.generationCount;
  }

  Population(City[] cities, int populationNum, float crossoverRate, float mutationRate) {
    this.cities = cities;
    this.populationNum = populationNum;
    this.population = new DNA[populationNum];
```

```
    this.crossoverRate = crossoverRate;
    this.mutationRate = mutationRate;
    println("Generation " + generationCount);
    println("----------------------------");
    for (int i = 0; i < populationNum; i++) {
      DNA dna = new DNA(this.cities, true);
      population[i] = dna;
    }
  }

  void calcFitness() {
    sumFitness = 0;
    sumFitness += population[0].calcFitness();
    shortestDistance = population[0].getDistance();
    bestFitness = population[0].getFitness();
    bestOne = population[0];
    for (int i = 1; i < populationNum; i++) {
      double currentFitness = population[i].calcFitness();
      sumFitness += currentFitness;
      if (currentFitness > bestOne.getFitness()) {
        shortestDistance = population[i].getDistance();
        bestFitness = currentFitness;
        bestOne = population[i];
      }
    }
    println("Shortest Distance: " + shortestDistance);
    println("Best Fitness: " + bestFitness);
    println();
  }

  void naturalSelection() {
    DNA[] nextGeneration = new DNA[populationNum];
    nextGeneration[0] = new DNA(bestOne.getGenes(),false);
    for (int i = 1; i < populationNum; i += 2) {
      DNA[] parents = new DNA[2];
      for (int j = 0; j < 2; j++) {
        double randomFitness = random((float)(sumFitness));
        int index = -1;
        while (randomFitness >= 0) {
          randomFitness -= population[++index].getFitness();
        }
        parents[j] = population[index];
      }
      final int k = floor(random(cityNum));
```

```
        final int j = floor(random(cityNum));
        final int start = min(k,j);
        final int end = max(k,j);
        DNA child1 = parents[0].crossover(parents[1], start, end, crossoverRate);
        DNA child2 = parents[1].crossover(parents[0], start, end, crossoverRate);
        child1 = child1.mutate(mutationRate);
        child2 = child2.mutate(mutationRate);
        nextGeneration[i] = child1;
        nextGeneration[i+1] = child2;
      }
      this.population = nextGeneration;
      println("Generation " + ++generationCount);
      println("----------------------------");
    }
}
```

Appendix B: Test Data

Mutation Rate: 10%

Generation	Distance										
	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**	**Trial 6**	**Trial 7**	**Trial 8**	**Trial 9**	**Trial 10**	**Average**
100	13670.39	15072.42	14699.53	15131.36	15093.73	15012.88	15422.73	15564.42	14464.69	15292.65	15083.82
200	11574.69	13243.37	12997.63	12166.31	13129.77	12839.30	13067.95	12040.79	12818.96	13109.86	12823.77
300	11044.67	12118.10	11265.39	10340.52	12332.35	11759.26	11877.83	10884.28	11967.60	10572.40	11457.53
400	10495.44	10871.41	10247.97	8597.67	11265.50	10807.02	11286.71	10240.72	11186.16	10487.75	10554.55
500	9587.52	10283.42	9871.63	8425.97	10343.02	9781.61	10635.80	10066.25	9973.97	10459.05	9982.30
600	9316.93	9843.23	9298.10	8271.31	9984.03	9528.79	9890.14	10002.12	9557.92	10320.56	9632.91
700	9139.08	9658.48	8409.04	8205.45	9648.57	9226.37	9430.47	9447.51	9250.00	9662.46	9215.37
800	8851.64	9423.22	8274.32	8012.77	9361.44	8850.16	8904.95	9374.65	8928.12	8558.02	8854.18
900	8420.34	9070.95	7951.70	7882.21	9238.64	8646.89	8555.24	9205.73	8699.92	8386.40	8626.41
1000	8236.82	8786.90	7925.73	7882.21	9233.55	8517.32	8384.53	9059.60	8466.85	8188.46	8493.91
1100	8216.78	8662.24	7854.05	7882.21	9063.23	8505.16	8358.27	8871.83	8298.74	8155.47	8405.69
1200	8186.44	8536.54	7840.64	7882.21	8908.34	8294.81	8353.18	8707.19	8214.17	8070.06	8311.90
1300	8149.03	8335.82	7818.19	7882.21	8819.43	8294.81	8301.55	8707.19	8148.43	7989.47	8255.23
1400	8149.03	8329.27	7796.69	7882.21	8819.43	8216.89	8252.71	8704.46	7918.65	7972.43	8210.30
1500	8149.03	8307.76	7796.69	7865.12	8722.55	8216.89	8252.71	8668.01	7913.13	7972.43	8190.59
1600	8055.44	8279.02	7777.03	7865.12	8694.27	8069.00	8244.88	8608.98	7905.97	7972.43	8157.41
1700	8032.98	8279.02	7777.03	7865.12	8615.25	8067.06	8217.52	8511.85	7905.97	7972.43	8134.58
1800	8028.35	8194.64	7777.03	7865.12	8615.25	8067.06	8217.52	8408.13	7886.65	7972.43	8111.54
1900	7963.18	8130.80	7777.03	7865.12	8583.68	8067.06	8207.99	8400.41	7883.59	7972.43	8098.68
2000	7963.18	8130.80	7777.03	7865.12	8582.18	8049.55	8207.99	8400.41	7883.59	7972.43	8096.57
2100	7771.75	8123.30	7777.03	7865.12	8582.18	8040.08	8193.65	8377.89	7836.55	7972.43	8085.36
2200	7771.75	8105.52	7777.03	7865.12	8582.18	8025.67	8193.65	8358.14	7836.55	7972.43	8079.59
2300	7758.06	8105.52	7777.03	7865.12	8423.45	8004.17	8186.50	8358.14	7836.55	7972.43	8058.77
2400	7758.06	8100.00	7777.03	7865.12	8416.15	7998.15	8186.50	8358.14	7797.38	7972.43	8052.32
2500	7758.06	8100.00	7777.03	7865.12	8416.15	7998.15	8186.50	8352.46	7795.92	7972.43	8051.53
2600	7758.06	8100.00	7777.03	7865.12	8391.39	7998.15	8186.50	8287.10	7774.42	7972.43	8039.12
2700	7758.06	8100.00	7777.03	7865.12	8391.39	7998.15	8186.50	8287.10	7774.42	7972.43	8039.12
2800	7705.32	8096.94	7777.03	7865.12	8391.39	7947.36	8186.50	8287.10	7774.42	7972.43	8033.14
2900	7586.65	8096.94	7777.03	7865.12	8249.00	7947.36	8186.50	8284.70	7756.55	7972.43	8015.07
3000	7563.69	8087.56	7759.02	7865.12	8225.51	7947.36	8186.50	8284.70	7748.34	7972.43	8008.50
3100	7544.37	8087.56	7631.30	7797.28	8185.67	7947.36	8172.83	8271.94	7748.34	7972.43	7979.41
3200	7544.37	8079.07	7631.30	7797.28	8137.17	7947.36	8172.83	8271.94	7748.34	7972.43	7973.08
3300	7544.37	8077.93	7631.30	7797.28	8137.17	7935.04	8172.83	8271.94	7748.34	7972.43	7971.58
3400	7544.37	8077.93	7606.95	7797.28	8074.53	7935.04	8172.83	8271.94	7748.34	7972.43	7961.92
3500	7544.37	8077.93	7606.95	7797.28	8074.53	7935.04	8172.83	8271.94	7748.34	7972.43	7961.92
3600	7544.37	8077.93	7606.95	7797.28	7933.59	7935.04	8172.83	8271.94	7748.34	7972.43	7946.26
3700	7544.37	8077.93	7602.04	7797.28	7844.25	7935.04	8172.83	8271.94	7748.34	7972.43	7935.79

3800	7544.37	8077.93	7602.04	7797.28	7798.38	7935.04	8172.83	8271.94	7748.34	7972.43	7930.69
3900	7544.37	8077.93	7544.37	7797.28	7606.95	7935.04	8172.83	8271.94	7748.34	7972.43	7903.01
4000	7544.37	8077.93	7544.37	7797.28	7602.04	7935.04	8172.83	8271.94	7748.34	7972.43	7902.47
4100	7544.37	8077.93	7544.37	7797.28	7544.37	7935.04	8172.83	8271.94	7748.34	7972.43	7896.06
4200	7544.37	8077.93	7544.37	7797.28	7544.37	7935.04	8172.83	8261.81	7748.34	7972.43	7894.93
4300	7544.37	8070.85	7544.37	7797.28	7544.37	7935.04	8172.83	8261.81	7748.34	7972.43	7894.15
4400	7544.37	8070.85	7544.37	7797.28	7544.37	7935.04	8172.83	8261.81	7748.34	7972.43	7894.15
4500	7544.37	8070.85	7544.37	7797.28	7544.37	7935.04	8172.83	8261.81	7748.34	7972.43	7894.15
4600	7544.37	8070.85	7544.37	7797.28	7544.37	7935.04	8172.83	8261.81	7748.34	7972.43	7894.15
4700	7544.37	8070.85	7544.37	7797.28	7544.37	7935.04	8172.83	8261.81	7748.34	7972.43	7894.15
4800	7544.37	8070.85	7544.37	7797.28	7544.37	7935.04	8172.83	8260.99	7748.34	7972.43	7894.06
4900	7544.37	8070.85	7544.37	7797.28	7544.37	7935.04	8172.83	8254.90	7748.34	7972.43	7893.38
5000	7544.37	8070.85	7544.37	7797.28	7544.37	7935.04	8172.83	8254.90	7748.34	7972.43	7893.38

Mutation Rate: 30%

Generation	Distance										
	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**	**Trial 6**	**Trial 7**	**Trial 8**	**Trial 9**	**Trial 10**	**Average**
100	15230.55	16145.14	15524.61	14920.34	14838.64	14530.04	16036.86	15006.18	16690.50	15507.14	15466.61
200	13460.11	13283.69	13681.03	14488.33	13717.43	13686.15	13195.66	13861.55	13832.97	13226.37	13663.69
300	12479.56	12314.49	11464.72	13493.56	12396.09	12116.19	11826.77	12669.06	11942.04	12367.69	12287.84
400	12119.17	11199.12	10697.88	11979.34	11434.29	11170.73	11211.41	11903.00	11371.30	11502.45	11385.50
500	11363.10	10403.98	9827.58	10989.09	10786.20	10152.89	10613.38	11243.33	10871.71	10997.65	10653.98
600	10595.14	9926.40	9354.96	10713.31	10374.18	10115.18	10050.03	11080.20	10382.10	10195.94	10243.59
700	10026.06	9747.97	9105.97	10501.29	10338.59	9545.57	9411.74	10588.83	9809.66	9990.32	9893.33
800	9887.30	9471.50	8768.55	10036.80	9819.41	8989.60	8774.76	9731.06	9619.31	9711.58	9435.84
900	9364.12	9351.03	8361.60	10036.68	9648.08	8598.49	8636.86	8825.02	9566.69	9711.58	9192.89
1000	9311.63	9330.57	7982.52	9691.35	9045.34	8496.14	8557.98	8664.85	9185.96	9626.65	8953.48
1100	9132.75	8864.20	7963.20	9388.58	8994.91	8477.04	8557.98	8575.30	9134.70	9368.65	8813.84
1200	8947.01	8817.37	7963.20	9276.81	8990.28	8436.24	8500.78	8468.77	8762.59	9296.56	8723.62
1300	8846.77	8704.68	7963.20	9069.93	8928.16	8436.24	8430.92	8468.77	8283.59	8968.11	8583.73
1400	8794.11	8460.52	7948.88	9032.94	8401.05	8352.25	8400.48	8377.25	8119.06	8941.08	8448.17
1500	8794.11	8312.68	7948.88	8885.40	8283.73	8324.22	8360.63	8377.25	8082.50	8848.60	8380.43
1600	8763.03	8289.42	7899.36	8885.40	8276.44	8297.86	8200.54	8211.08	8061.00	8534.13	8295.03
1700	8679.50	8241.53	7875.77	8745.72	8215.63	8257.02	8163.85	8116.07	8003.74	8480.76	8233.34
1800	8635.26	8230.62	7875.77	8665.44	8205.68	8257.02	8074.04	8116.07	7959.80	8435.45	8202.21
1900	8564.88	8230.49	7875.77	8557.31	8205.68	8195.26	8073.77	8090.43	7931.17	8435.45	8177.26
2000	8564.88	8230.49	7875.77	8504.70	8111.99	8161.21	7956.68	8090.43	7918.44	8435.45	8142.80
2100	8510.73	8160.40	7875.77	8496.88	8111.99	8082.20	7956.68	8090.43	7918.44	8382.94	8119.53
2200	8503.81	8156.52	7875.77	8496.88	8111.99	8082.20	7956.68	8090.43	7906.30	8382.94	8117.75
2300	8482.31	8156.52	7875.77	8496.88	8107.36	8082.20	7921.96	7986.51	7906.30	8382.94	8101.83
2400	8482.31	8156.52	7875.77	8496.88	8107.36	8082.20	7859.91	7911.73	7880.66	8382.94	8083.78
2500	8482.31	8148.73	7875.77	8459.35	8107.36	8082.20	7859.91	7911.73	7880.66	8382.94	8078.74
2600	8482.31	8055.54	7875.77	8459.35	8107.36	7914.19	7859.91	7900.69	7761.75	8382.94	8035.28
2700	8482.31	8055.54	7875.77	8406.34	8107.36	7864.56	7859.91	7900.69	7761.75	8382.94	8023.87
2800	8482.31	8055.54	7875.77	8331.70	8107.36	7744.65	7859.91	7848.07	7761.75	8355.83	7993.40
2900	8482.31	8040.10	7875.77	8331.70	8085.86	7686.98	7859.91	7797.28	7718.71	8355.83	7972.46
3000	8482.31	8040.10	7875.77	8331.70	8085.86	7686.98	7859.91	7797.28	7708.63	8354.22	7971.16
3100	8482.31	8001.40	7875.77	8323.49	8085.86	7682.36	7859.91	7797.28	7646.25	8279.06	7950.15
3200	8482.31	8001.40	7875.77	8323.49	8085.86	7682.36	7859.91	7797.28	7641.62	8279.06	7949.64
3300	8482.31	8001.40	7875.77	8323.49	8085.86	7682.36	7859.91	7797.28	7619.15	8279.06	7947.14
3400	8482.31	8001.40	7875.77	8323.49	8085.86	7682.36	7859.91	7797.28	7619.15	8279.06	7947.14
3500	8424.44	8001.40	7875.77	8323.49	8085.86	7682.36	7859.91	7797.28	7619.15	8199.68	7938.32
3600	8424.44	8001.40	7875.77	8323.49	8085.86	7682.36	7859.91	7797.28	7619.15	8199.68	7938.32
3700	8424.44	7997.52	7875.77	8315.99	8085.86	7682.36	7859.91	7797.28	7544.37	8199.68	7928.75
3800	8424.44	7894.30	7875.77	8315.99	8085.86	7682.36	7859.91	7797.28	7544.37	8199.68	7917.28

3900	8424.44	7894.30	7875.77	8315.99	8085.86	7682.36	7859.91	7797.28	7544.37	8199.68	7917.28
4000	8424.44	7894.30	7875.77	8315.99	8085.86	7682.36	7859.91	7797.28	7544.37	8199.68	7917.28
4100	8416.94	7894.30	7875.77	8315.99	8070.99	7682.36	7859.91	7797.28	7544.37	8199.68	7915.63
4200	8416.94	7894.30	7875.77	8315.99	8070.99	7544.37	7859.91	7797.28	7544.37	8199.68	7900.30
4300	8416.94	7894.30	7875.77	8315.99	8070.99	7544.37	7859.91	7797.28	7544.37	8199.68	7900.30
4400	8416.94	7894.30	7875.77	8315.99	8070.99	7544.37	7859.91	7797.28	7544.37	8199.68	7900.30
4500	8401.89	7894.30	7848.82	8315.99	8070.99	7544.37	7859.91	7797.28	7544.37	8199.68	7897.30
4600	8401.89	7894.30	7848.82	8315.99	8070.99	7544.37	7859.91	7797.28	7544.37	8192.19	7896.47
4700	8401.89	7894.30	7848.82	8315.99	8070.99	7544.37	7859.91	7797.28	7544.37	8192.19	7896.47
4800	8401.89	7894.30	7848.82	8315.99	8070.99	7544.37	7859.91	7797.28	7544.37	8192.19	7896.47
4900	8401.89	7894.30	7848.82	8315.99	8070.99	7544.37	7859.91	7797.28	7544.37	8192.19	7896.47
5000	8401.89	7894.30	7848.82	8315.99	8070.99	7544.37	7859.91	7797.28	7544.37	8192.19	7896.47

Mutation Rate: 50%

Generation	Distance										
	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**	**Trial 6**	**Trial 7**	**Trial 8**	**Trial 9**	**Trial 10**	**Average**
100	14960.33	15354.65	15581.87	14380.86	15897.18	16384.96	15818.66	14491.51	15626.54	17008.73	15616.11
200	13129.60	13775.06	12952.95	12944.78	13697.61	13010.16	13880.95	12345.61	12842.50	12961.68	13156.81
300	12117.55	13094.10	12161.91	10993.00	12661.15	12359.37	12149.16	12050.68	11891.02	11953.79	12146.02
400	11149.08	12127.80	11610.34	10700.50	11740.30	11889.81	11268.26	11745.48	11096.82	10965.41	11460.53
500	10428.65	11525.57	10784.16	10378.40	11168.93	11601.62	10697.83	11191.34	10730.51	10508.84	10954.13
600	10047.61	10758.68	10356.75	9992.83	10668.59	11350.82	10488.57	10914.13	10499.38	9975.60	10556.15
700	9082.44	10558.15	10098.33	9708.97	10128.09	10891.72	10141.91	10329.18	10155.55	9752.45	10196.04
800	8753.25	10327.03	9600.44	9664.79	9485.70	10464.00	9732.70	9806.87	9732.53	9484.42	9810.94
900	8732.52	9998.48	9502.32	9085.10	9131.26	10338.23	9091.95	9698.64	9497.67	8804.78	9460.94
1000	8365.98	9460.39	9426.19	8991.01	8942.17	10047.63	8631.81	9437.40	9201.69	8636.36	9197.18
1100	8365.98	9380.76	9216.87	8882.88	8853.78	9321.66	8567.31	9412.61	9166.55	8377.58	9020.00
1200	8365.98	9265.57	9205.38	8787.80	8720.31	9165.11	8548.02	9343.07	8826.21	8259.32	8902.31
1300	8321.54	9265.57	9157.67	8704.19	8668.45	9133.07	8514.05	9292.35	8750.29	8141.01	8847.40
1400	8274.68	9033.14	8989.38	8703.77	8564.53	9133.07	8455.43	9174.90	8750.29	8108.72	8768.14
1500	8274.68	8955.01	8989.38	8655.00	8489.21	8955.72	8433.06	8912.24	8660.49	8108.46	8684.29
1600	8193.27	8955.01	8989.38	8651.95	8483.10	8918.95	8425.89	8774.99	8581.87	8108.46	8654.40
1700	8193.27	8875.95	8905.93	8595.70	8463.61	8689.03	8116.37	8761.51	8309.66	8108.46	8536.25
1800	8193.27	8660.36	8860.60	8487.06	8323.24	8604.18	7981.83	8708.45	8064.78	8108.46	8422.11
1900	8193.27	8511.84	8843.09	8487.06	8310.27	8438.87	7930.59	8708.45	8045.19	8103.67	8375.45
2000	8193.27	8492.68	8758.88	8415.16	8224.53	8286.81	7837.11	8671.21	7977.94	8068.95	8303.70
2100	8193.27	8252.49	8636.17	8415.16	8126.33	8258.04	7837.11	8617.78	7977.94	8051.24	8241.36
2200	8171.96	8205.47	8634.07	8415.16	8050.97	8185.53	7795.08	8582.60	7977.94	8031.24	8208.67
2300	8154.25	8174.56	8514.43	8411.95	8026.21	8127.47	7777.30	8582.60	7950.56	8031.24	8177.37
2400	8154.25	8110.10	8271.99	8400.33	8019.07	8127.47	7777.30	8451.95	7950.56	8031.24	8126.67
2500	8154.25	8110.10	7892.79	8393.26	8019.07	8100.36	7777.30	8451.95	7950.56	8031.24	8080.74
2600	8154.25	8110.10	7840.73	8393.26	8019.07	8100.36	7652.41	8451.95	7950.56	8031.24	8061.07
2700	8154.25	8110.10	7821.41	8312.79	8019.07	8075.29	7625.40	8451.95	7950.56	8031.24	8044.20
2800	8154.25	8110.10	7821.41	8293.69	8019.07	8075.29	7625.40	8451.95	7950.56	8008.31	8039.53
2900	8142.81	8083.84	7821.41	8115.02	8019.07	8075.29	7598.44	8451.95	7950.56	8008.31	8013.77
3000	8142.81	8083.84	7778.06	8115.02	8019.07	8075.29	7598.44	8451.95	7950.56	8008.31	8008.95
3100	8142.81	8083.84	7778.06	8032.93	8019.07	8075.29	7598.44	8451.95	7931.46	8008.31	7997.71
3200	8142.81	8044.67	7778.06	8032.93	8019.07	8075.29	7598.44	8451.95	7931.46	8008.31	7993.35
3300	8142.81	8044.67	7778.06	8028.63	8019.07	8075.29	7598.44	8432.29	7931.46	8008.31	7990.69
3400	8142.81	8044.67	7778.06	8028.63	8019.07	8068.15	7598.44	8425.15	7856.67	8008.31	7980.80
3500	8142.81	8044.67	7778.06	8013.93	8019.07	8068.15	7598.44	8425.15	7856.67	8008.31	7979.16
3600	8017.51	8044.67	7778.06	8013.93	8019.07	8068.15	7598.44	8425.15	7856.67	8008.31	7979.16
3700	8017.51	8044.67	7778.06	8013.93	8019.07	8068.15	7598.44	8425.15	7856.67	8008.31	7979.16
3800	8015.00	8044.67	7778.06	8013.93	8019.07	8068.15	7598.44	8425.15	7856.67	8008.31	7979.16

3900	7993.45	8044.67	7778.06	8013.93	8019.07	8068.15	7598.44	8425.15	7856.67	8008.31	7979.16
4000	7988.82	8044.67	7778.06	8013.93	8019.07	8058.78	7598.44	8425.15	7856.67	8008.31	7978.12
4100	7967.59	8044.67	7778.06	8013.93	8019.07	8058.78	7598.44	8425.15	7856.67	8008.31	7978.12
4200	7967.59	8033.57	7778.06	8013.93	8014.08	8058.78	7598.44	8425.15	7856.67	8008.31	7976.33
4300	7967.59	8033.57	7778.06	8013.93	8014.08	8058.78	7598.44	8425.15	7830.38	8008.31	7973.41
4400	7945.12	8033.57	7778.06	8013.93	8014.08	8053.25	7598.44	8425.15	7830.38	8008.31	7972.80
4500	7945.12	8033.57	7778.06	8013.93	8014.08	8053.25	7598.44	8425.15	7817.03	8008.31	7971.31
4600	7945.12	8033.57	7778.06	8013.93	8014.08	8053.25	7598.44	8425.15	7817.03	8008.31	7971.31
4700	7945.12	8033.57	7778.06	8013.93	8014.08	8053.25	7598.44	8425.15	7800.49	8008.31	7969.48
4800	7945.12	8033.57	7778.06	8013.93	8014.08	8048.26	7598.44	8425.15	7800.49	8008.31	7968.92
4900	7945.12	8033.57	7778.06	8013.93	8014.08	8048.26	7598.44	8302.29	7800.49	8008.31	7955.27
5000	7945.12	8033.57	7778.06	8013.93	8014.08	8048.26	7598.44	8302.29	7800.49	8008.31	7955.27

Mutation Rate: 70%

Generation	Distance										
	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**	**Trial 6**	**Trial 7**	**Trial 8**	**Trial 9**	**Trial 10**	**Average**
100	15472.87	15230.08	14824.44	17689.02	15838.91	16931.74	16185.52	17408.58	15455.04	15476.99	16115.59
200	14026.44	14468.09	12703.03	15155.55	13807.87	14531.15	13824.43	15924.22	14214.24	12502.72	14125.70
300	12506.09	13395.10	11205.42	13602.76	12555.91	13003.91	12075.56	14354.54	13264.87	11102.79	12728.98
400	12073.91	11164.11	10776.39	11506.67	11947.69	12451.38	11380.03	12808.14	12395.55	9798.67	11580.96
500	10687.80	10787.91	10478.44	11079.75	11689.97	12311.80	10323.42	11353.70	10954.65	9503.08	10942.53
600	10177.23	10295.67	10318.26	10495.24	11207.98	11942.92	10007.99	10986.22	10685.02	9452.82	10599.12
700	9604.44	10020.28	9699.14	10400.23	9966.44	11729.93	9989.43	10557.53	10582.43	9167.91	10234.81
800	9558.82	9325.01	9377.48	10212.58	9547.81	11051.09	9903.88	10038.70	10265.29	8893.73	9846.18
900	9458.78	9038.23	9377.48	9864.47	9328.59	10261.82	9884.66	9903.42	10202.86	8563.03	9602.73
1000	9320.39	9030.91	9377.48	9834.50	9095.35	9955.85	9726.48	9646.65	9623.61	8511.42	9422.47
1100	9127.88	9030.91	9042.36	9646.02	9044.77	9885.66	9392.69	9493.38	9616.11	8461.42	9290.37
1200	8904.64	9030.91	8977.01	9568.05	8988.62	9630.66	9163.00	9315.34	9440.30	8439.83	9172.64
1300	8460.56	9004.47	8940.96	9500.96	8870.42	9387.62	9087.38	8846.43	9247.68	8439.83	9036.20
1400	8459.63	8995.18	8771.97	9500.96	8829.93	9288.36	9019.11	8632.22	9028.68	8187.54	8917.11
1500	8444.25	8995.18	8730.32	9495.41	8770.98	9288.06	8991.07	8606.72	8879.20	8187.54	8882.72
1600	8068.81	8833.46	8597.89	9379.97	8567.96	9143.14	8892.08	8606.72	8879.20	8187.54	8787.55
1700	8053.76	8695.13	8503.58	9278.72	8567.45	8935.27	8892.08	8600.04	8741.87	8117.96	8703.57
1800	8053.76	8660.05	8388.30	9273.19	8290.11	8897.95	8853.71	8550.29	8651.47	8100.49	8629.51
1900	8042.67	8601.29	8207.39	9199.47	8264.72	8858.49	8617.27	8550.29	8600.82	8100.49	8555.58
2000	7949.85	8508.60	7968.13	9199.47	8142.01	8827.67	8602.21	8527.82	8600.82	8060.82	8493.06
2100	7883.54	8394.39	7954.49	9199.47	8062.00	8793.86	8592.13	8527.82	8583.11	8020.84	8458.68
2200	7883.54	8394.39	7921.66	9001.98	8049.42	8709.95	8592.13	8438.73	8438.61	8020.84	8396.41
2300	7883.54	8375.29	7921.66	9001.98	7926.46	8652.28	8591.54	8438.73	8416.14	7943.16	8363.03
2400	7872.51	8375.29	7865.57	8952.49	7827.52	8652.28	8550.71	8402.63	8362.50	7943.16	8325.79
2500	7872.51	8375.29	7865.57	8952.49	7827.52	8652.28	8485.03	8402.63	8287.71	7943.16	8310.19
2600	7872.51	8370.66	7843.11	8952.49	7827.52	8652.28	8344.28	8402.63	8287.71	7943.16	8291.54
2700	7872.51	8276.48	7843.11	8889.51	7827.52	8652.28	8272.61	8351.88	8238.44	7943.16	8255.00
2800	7842.78	8179.76	7842.81	8774.11	7819.25	8646.59	8255.52	8351.88	8238.44	7943.16	8227.95
2900	7833.48	8179.76	7842.81	8710.50	7819.25	8646.59	8255.52	8351.88	8172.10	7943.16	8213.51
3000	7798.77	8111.93	7842.81	8630.78	7819.25	8646.59	8255.52	8351.88	8172.10	7943.16	8197.11
3100	7798.77	8111.93	7842.81	8542.43	7819.25	8646.59	8255.52	8351.88	8172.10	7943.16	8187.30
3200	7798.77	7907.25	7842.81	8540.45	7819.25	8434.69	8255.52	8351.88	8164.38	7943.16	8139.93
3300	7798.77	7907.25	7842.81	8489.26	7819.25	8122.13	8255.52	8351.88	8164.38	7943.16	8099.51
3400	7793.24	7907.25	7842.81	8413.69	7819.25	8122.13	8255.52	8351.88	8123.54	7943.16	8086.58
3500	7793.24	7907.25	7842.81	8218.57	7819.25	8122.13	8255.52	8351.88	8123.54	7888.96	8058.88
3600	7793.24	7907.25	7842.81	8218.57	7819.25	8122.13	8255.52	8351.88	8123.54	7888.96	8058.88
3700	7793.24	7907.25	7842.81	8218.57	7819.25	8122.13	8255.52	8351.88	8123.54	7888.96	8058.88
3800	7793.24	7907.25	7842.81	8218.57	7819.25	8122.13	8222.70	8351.88	8123.54	7888.96	8055.23

3900	7793.24	7907.25	7842.81	8139.67	7819.25	8122.13	8169.95	8351.88	8123.54	7888.96	8040.60
4000	7793.24	7907.25	7842.81	8139.67	7819.25	8122.13	8168.49	8351.88	8123.54	7888.96	8040.44
4100	7793.24	7907.25	7842.81	8139.67	7819.25	8122.13	8168.49	8351.88	8123.54	7888.96	8040.44
4200	7793.24	7907.25	7842.81	8139.67	7819.25	8122.13	8168.49	8351.88	8123.54	7888.96	8040.44
4300	7793.24	7796.66	7821.30	8135.51	7819.25	8122.13	8146.99	8351.88	8123.54	7888.96	8022.91
4400	7793.24	7796.66	7821.30	8135.51	7819.25	8122.13	8146.99	8351.88	8123.54	7888.96	8022.91
4500	7793.24	7796.66	7821.30	8135.51	7819.25	8122.13	8146.99	8351.88	8123.54	7880.38	8021.96
4600	7793.24	7777.33	7821.30	8128.44	7819.25	8122.13	8146.99	8351.88	8123.54	7880.38	8019.03
4700	7793.24	7777.33	7821.30	8128.44	7819.25	8117.42	8146.99	8351.88	8123.54	7880.38	8018.50
4800	7793.24	7777.33	7821.30	8128.44	7819.25	8117.42	8146.99	8351.88	8123.54	7880.38	8018.50
4900	7793.24	7777.33	7821.30	8126.33	7819.25	8041.55	8146.99	8351.88	8123.54	7880.38	8009.84
5000	7793.24	7777.33	7821.30	8126.33	7819.25	8041.55	8146.99	8351.88	8123.54	7880.38	8009.84

Mutation Rate: 90%

Generation	Distance										
	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**	**Trial 6**	**Trial 7**	**Trial 8**	**Trial 9**	**Trial 10**	**Average**
100	16153.37	15806.43	17021.44	16660.23	17341.99	15281.15	16188.90	14905.23	17430.32	15546.95	16242.51
200	14491.63	13058.92	14875.63	15403.94	14395.98	13363.76	14186.58	13735.44	13603.02	13625.12	14027.60
300	13562.89	12770.12	13183.17	13258.33	11571.80	13090.01	13317.08	13096.93	12100.40	12331.36	12746.58
400	12146.93	12361.49	11115.40	12012.26	11477.14	12757.69	11807.42	12768.48	10824.09	11127.03	11805.67
500	10695.53	11677.09	10254.56	11563.50	10706.25	12703.07	11580.09	11745.96	10504.48	10753.63	11276.52
600	10463.36	11024.50	9708.02	11148.15	10559.81	12146.52	11422.38	10984.16	10386.15	10555.65	10881.70
700	10218.35	10701.48	9600.44	10870.55	10197.34	11800.61	11309.26	10377.11	10008.39	10414.84	10586.67
800	10134.27	10595.09	9547.82	10573.88	10197.34	11634.66	11081.27	10009.50	9851.22	10376.60	10429.71
900	10115.77	10592.52	9346.59	10573.88	10130.58	11399.18	10963.42	9983.24	9767.96	10150.98	10323.15
1000	9905.73	10446.00	9169.47	10460.76	9988.46	11136.60	10853.73	9868.19	9380.99	10111.20	10157.27
1100	9765.42	9993.24	9028.23	9617.56	9721.58	10996.90	10742.89	9683.09	9336.27	9926.96	9894.08
1200	9627.90	9885.55	8950.89	9617.56	9613.78	10665.55	10226.51	9334.02	9119.73	9679.13	9676.97
1300	9291.17	9756.46	8950.89	9444.21	9507.06	10377.57	9973.83	9311.56	9108.62	9411.84	9538.00
1400	9198.97	9612.67	8936.34	9404.64	9353.28	10281.82	9856.39	9103.39	9011.96	9322.99	9431.50
1500	8892.48	9553.02	8783.78	9400.50	9299.21	10196.70	9659.78	9103.39	8964.67	9237.37	9355.38
1600	8724.37	9483.80	8783.78	9242.50	9292.60	10030.64	9510.58	8943.93	8823.56	9232.26	9260.41
1700	8716.54	9475.02	8782.73	9230.20	9262.82	10027.91	9348.61	8910.67	8795.44	9183.88	9224.14
1800	8567.53	9283.07	8782.73	9099.75	9233.66	10027.91	9313.38	8910.37	8689.24	9178.79	9168.76
1900	8532.33	9228.11	8709.88	8863.75	9071.09	10027.91	9304.51	8910.37	8580.99	9125.92	9091.39
2000	8480.52	9228.11	8560.59	8858.12	8969.64	9883.93	8888.48	8905.77	8479.24	9125.92	8988.87
2100	8480.52	9125.55	8501.37	8714.76	8969.64	9761.33	8888.48	8905.77	8433.62	9125.92	8936.27
2200	8480.52	8953.58	8501.37	8671.87	8969.64	9680.94	8779.22	8706.21	8258.66	9125.92	8849.71
2300	8445.81	8943.51	8482.95	8671.87	8895.73	9680.94	8779.22	8693.63	8258.66	9125.92	8836.93
2400	8151.70	8943.51	8482.95	8514.61	8872.35	9571.89	8779.22	8693.63	8082.85	9125.92	8785.21
2500	8151.70	8876.00	8425.28	8424.06	8869.11	9545.68	8779.16	8614.02	8082.85	9125.92	8749.12
2600	8151.70	8832.19	8425.28	8423.93	8869.11	9391.63	8745.22	8588.55	8082.85	9032.87	8710.18
2700	8151.70	8825.70	8425.28	8423.93	8869.11	9391.63	8650.25	8588.55	8082.85	8722.19	8664.39
2800	8151.70	8735.26	8425.28	8423.93	8869.11	9157.68	8606.57	8542.87	8082.85	8702.19	8616.19
2900	8150.24	8720.94	8425.28	8418.13	8869.11	9157.68	8586.43	8484.70	8082.85	8702.19	8605.26
3000	8150.24	8580.36	8349.41	8367.36	8869.11	9120.68	8567.12	8410.62	8082.85	8702.19	8561.08
3100	8128.74	8580.36	8349.41	8353.04	8771.00	8864.29	8567.12	8406.08	8082.85	8702.19	8519.59
3200	8128.74	8364.11	8349.41	8338.97	8771.00	8726.12	8567.12	8406.08	8082.85	8702.19	8478.65
3300	8115.36	8364.11	8349.41	8271.24	8733.74	8690.01	8567.12	8406.08	8082.85	8702.19	8462.97
3400	8115.36	8364.11	8349.41	8271.24	8733.74	8644.45	8567.12	8380.44	8082.85	8649.25	8449.18
3500	8115.36	8364.11	8349.41	8271.24	8733.74	8598.35	8567.12	8359.33	8062.71	8569.24	8430.58
3600	8115.36	8364.11	8196.94	8252.12	8713.58	8598.35	8555.76	8359.33	8062.71	8504.40	8400.81
3700	8115.36	8326.94	8196.94	8199.10	8713.58	8598.35	8528.66	8359.33	8062.71	8504.40	8387.78
3800	8115.36	8326.94	8196.94	7941.83	8713.58	8567.16	8528.66	8359.33	8062.71	8372.85	8341.11

3900	8115.36	8315.91	8196.94	7941.83	8713.58	8567.16	8528.66	8359.33	8062.71	8372.85	8339.88
4000	8115.36	8315.91	8196.94	7941.83	8713.58	8567.16	8519.36	8359.33	8062.71	8359.13	8337.33
4100	8115.36	8315.91	8196.94	7941.83	8713.58	8567.16	8519.36	8359.33	8062.71	8334.78	8334.62
4200	8115.36	8315.91	8196.94	7941.83	8713.58	8418.04	8457.45	8359.33	8062.71	8334.78	8311.17
4300	8115.36	8315.91	8196.94	7941.83	8713.58	8418.04	8457.45	8359.33	8062.71	8255.76	8302.39
4400	8115.36	8315.91	8196.94	7941.83	8713.58	8418.04	8354.47	8359.05	8062.71	8255.76	8290.92
4500	8115.36	8315.91	8196.94	7930.92	8713.58	8412.51	8354.47	8333.96	8062.71	8255.76	8286.31
4600	8115.36	8315.91	8196.94	7930.92	8713.58	8412.51	8354.47	8333.96	8062.71	8132.85	8272.65
4700	8115.36	8315.91	8196.94	7930.92	8541.92	8412.51	8354.47	8171.22	8062.71	8130.99	8235.29
4800	8115.36	8315.91	8196.94	7911.60	8541.92	8412.51	8354.47	8171.22	8062.71	8130.99	8233.14
4900	8115.36	8315.91	8196.94	7911.60	8541.92	8412.51	8354.47	8146.79	8062.71	8124.59	8229.72
5000	8068.19	8315.91	8196.94	7911.60	8541.92	8412.51	8228.81	8075.13	8062.71	8124.59	8207.79