King Saud University
College of Computer and Information Sciences
Information Technology Department
IT422: Intelligent Systems

Genetic Algorithms

Programming Assignment Report

Prepared by :-

Group:	#7		
Group Members:	Raghad Alsheddi	31944	
	Haifa Alshathri	31944	
	Ruba Alotaibi	31944	
	Ghaida Alkharashi	32946	

Supervised by :-
I. Abeer Aldrees

1st Semester 1439/40 H

Table of Contents

1. SOLUTION REPRESENTATION

Genetic algorithms solution will begin with three input (population size, crossover rate, and mutation rate). It will generate initial random population (first generation) by filling population ArrayList, each population has many chromosomes (Bag) which consist of genes with strings of length 5 which is the number of items that randomly generated (1s and 0s). After initiate population, we evaluate fitness of each population member. Then apply crossover and mutation function by selecting two chromosomes from population using roulette wheel selection to generate new population. After that it compares the fitness function of parents and children to do a replacement. This process will continue until there is no improvement in the population for X iterations.

No.	Item	Weight (W)	Survival Points (SP)
1	Sleeping bag	5	50
2	Rope	4	40
3	Pocket Knife	1	15
4	Torch	2	20
5	Bottle	3	25

2. FITNESS FUNCTION

The fitness function evaluates how close a given solution is to the optimum solution of the desired problem. It determines how fit a solution is by summing the corresponding weights and survival points (separately) for each population member one by one. It then compares the population member's total weight to the backpack capacity. If the backpack capacity has been exceeded by the population member's total weight, then the fitness value is set to 0. Otherwise, the population member's corresponding total survival points is set as the fitness value and returned.

```
private double calFitness(String gene) {
   double total_weight = 0;
   double total_value = 0;
   double fitness_value = 0;
   double difference = 0;
   char c = '0';

   for(int j = 0; j < itemsNum; j ++) {
      c = gene.charAt(j);

      if(c == '1') {
         total_weight = total_weight + weightItems.get(j);
         total_value = total_value + spItems.get(j);
      }
   }
   difference = backpackCapacity - total_weight;
   if(difference >= 0) {

      fitness_value = total_value;
   }

   return fitness_value;
}
```

3. GENETIC OPERATORS

3.1 Crossover

Crossover is a genetic operator used to combine the genetic information of two parents to generate new offspring. Not all genes are chosen for cross over. To decide crossover or not is depending on the random number and the crossover rate. If the random number is greater than the crossover rates the crossover will not occur. Otherwise, the crossover will occur by choosing two genes randomly then choose a random number as a point to exchange all bits to one side of the point in both genes. At the end it will add it to the new population.

```
private void crossoverGenes(int firstParent, int secondParent) {

   String firstChild;
   String secondChild;


   double rand_crossover = Math.random();
   if(rand_crossover <= crossoveRate) {
      crossoverCount = crossoverCount + 1;
      Random generator = new Random();
      int cross_point = generator.nextInt(itemsNum) + 1;

      firstChild = population.get(firstParent).substring(0, cross_point) + population.get(secondParent).substring(cross_point);
      secondChild = population.get(secondParent).substring(0, cross_point) + population.get(firstParent).substring(cross_point);

      breedPopulation.add(firstChild);
      breedPopulation.add(secondChild);
      mutateGene();
      replacment(firstParent,secondParent);


   }
   else {
      cloneCount = cloneCount + 1;
      breedPopulation.add(population.get(firstParent));
      breedPopulation.add(population.get(secondParent));
      mutateGene();
   }



}
```

3.2 Mutation

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. To get a new solution, we use mutation as a small random tweak in the chromosome. Decide if the mutation is being used or not is depending on a random number and the mutation rate. If the random number is greater than the mutation rates the mutation will not occur. However, the mutation will occur if the random number is greater than or equal the mutation rates.

```
private void mutateGene() {

   double cc = Math.random();

   if(cc <= mutationRate) {

      mutation = true;
      String mut_gene;
      String new_mut_gene;
      Random generator = new Random();
      int mut_point = 0;
      double which_gene = Math.random() * 100;


      if(which_gene <= 50) {
         checkIndex = true;
         mut_gene = breedPopulation.get(breedPopulation.size() - 1);
         mut_point = generator.nextInt(itemsNum);



         if(mut_gene.substring(mut_point, mut_point + 1).equals("1")) {
            new_mut_gene = mut_gene.substring(0, mut_point) + "0" + mut_gene.substring(mut_point+1);
            breedPopulation.set(breedPopulation.size() - 1, new_mut_gene);
         }

         else

         if(mut_gene.substring(mut_point, mut_point + 1).equals("0")) {
            new_mut_gene = mut_gene.substring(0, mut_point) + "1" + mut_gene.substring(mut_point+1);
            breedPopulation.set(breedPopulation.size() - 1, new_mut_gene);
         }

      }


      if(which_gene > 50) {
         checkIndex = false;
         mut_gene = breedPopulation.get(breedPopulation.size() - 2);
         mut_point = generator.nextInt(itemsNum);
         if(mut_gene.substring(mut_point, mut_point + 1).equals("1")) {
            new_mut_gene = mut_gene.substring(0, mut_point) + "0" + mut_gene.substring(mut_point+1);
            breedPopulation.set(breedPopulation.size() - 2, new_mut_gene);
         }

         else

         if(mut_gene.substring(mut_point, mut_point + 1).equals("0")) {
            new_mut_gene = mut_gene.substring(0, mut_point) + "1" + mut_gene.substring(mut_point+1);
            breedPopulation.set(breedPopulation.size() - 2, new_mut_gene);
         }
      }

   }
}
```

3.3 Selection (Roulette wheel selection)

We used roulette wheel selection method by generating random number between 0 and total fitness, then used it to select gene based on the fitness level. If the fitness greater than or equal the random number, it will return the index of the gene. Otherwise, it will subtract the fitness from the random number.

```
private int selection() {

   double rand = Math.random() * totalFitness;

   for(int i = 0; i < populationSize; i++) {

      if(rand <= fitness.get(i)) {
         return i;
      }
      rand = rand - fitness.get(i);
   }

   return 0;
}
```

3.4 Replacement

The replacement function will compare the fitness functions of the parents and the children. Then it will choose the best fitness value and will return it to the population. This method will be implemented after the mutation.

```
private void replacment(int p1, int p2) {


   double parent1 = calFitness(population.get(p1));
   double parent2 = calFitness(population.get(p2));

   String c1 = breedPopulation.get(breedPopulation.size() - 2);
   String c2 = breedPopulation.get(breedPopulation.size() - 1);

   double child1 = calFitness(c1);
   double child2 = calFitness(c2);

   double F1 = parent1+parent2;
   double F2 = child1+child2;

   if(F1>F2){

      if(!checkIndex)
         breedPopulation.set(breedPopulation.size() - 2, population.get(p1));
      else
         breedPopulation.set(breedPopulation.size() - 1, population.get(p2));

   }
   else
   {
      if(!checkIndex)
         breedPopulation.set(breedPopulation.size() - 2, c1);
      else
         breedPopulation.set(breedPopulation.size() - 1, c2);

   }

   checkIndex = false;
}
```

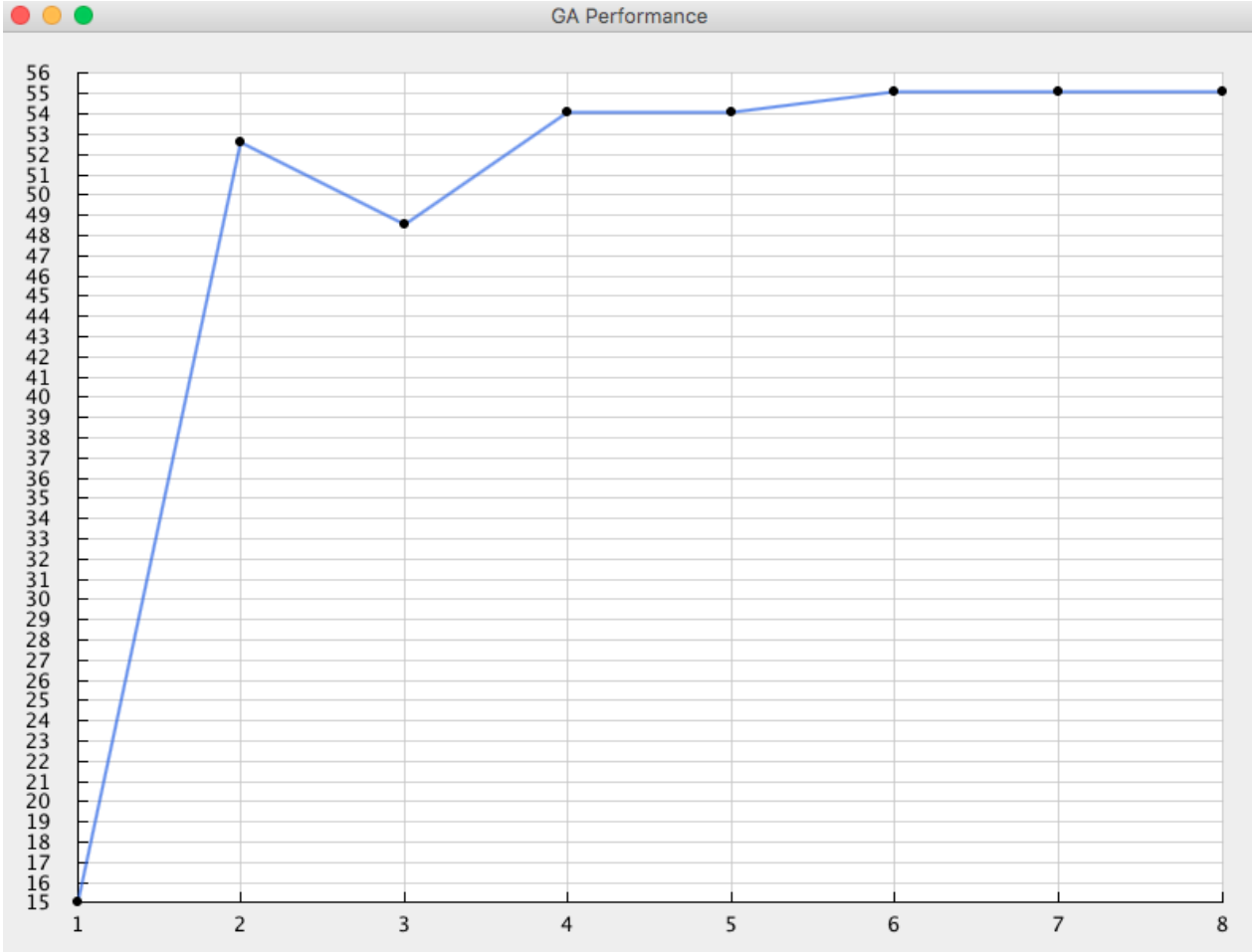
3.5 Termination condition

Termination code determines when a GA run will be ended. The termination condition is when there has been no improvement in the population for X iterations. Depending on the average fitness, if repeated 3 consecutive times the program will stop.

```
private void terminationCode() {

   for(int i = 1; i < this.generationsMax; i++) {

      if((this.generationsMax > 4) && (i > 4)) {

         double a = this.averageFitness.get(i - 1);
         double b = this.averageFitness.get(i - 2);
         double c = this.averageFitness.get(i - 3);

         if(a == b && b == c) {
            System.out.println("\nNo improvement in the population!!");
            generationsMax = i;
            break;
         }
      }
```

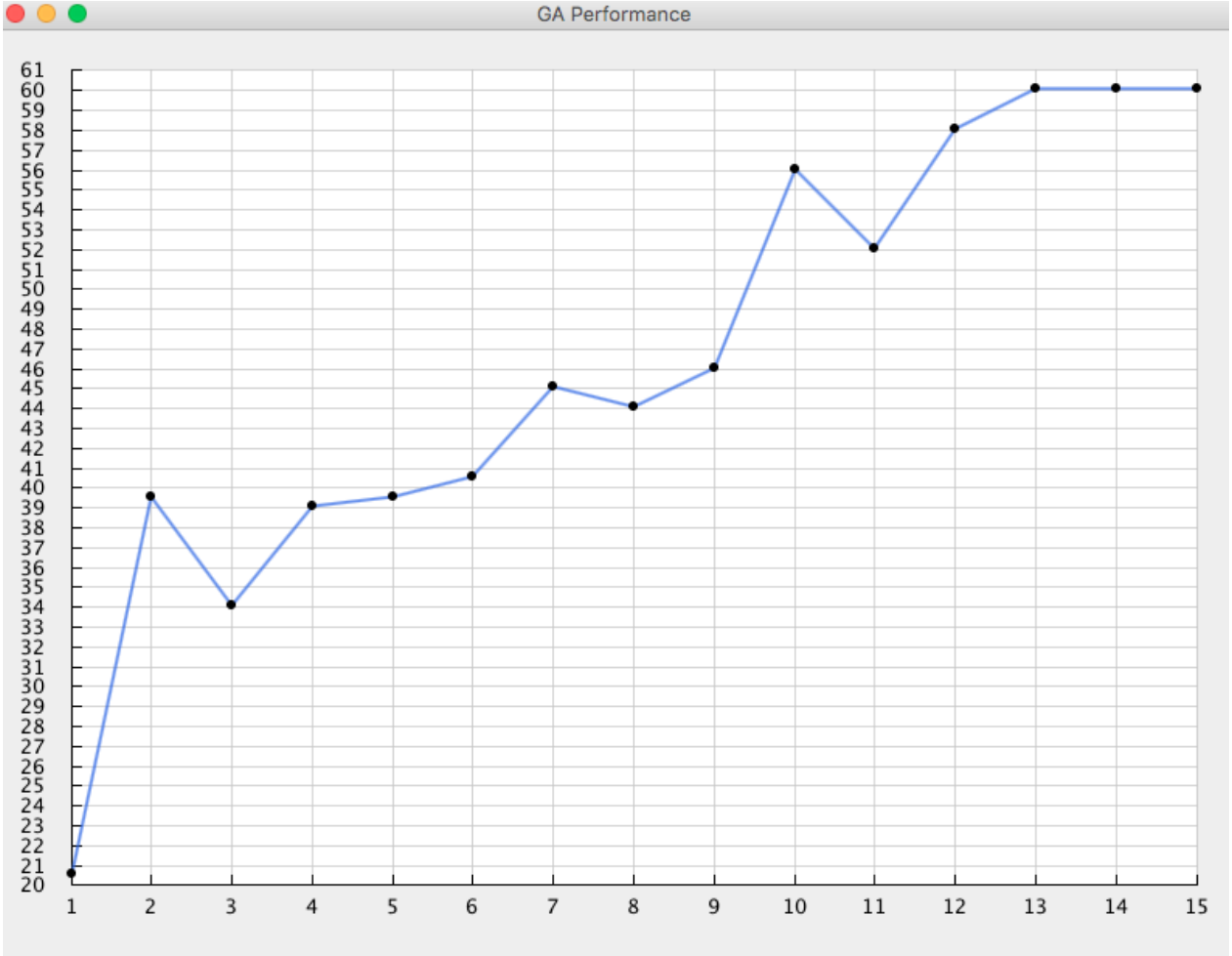
4. RESULT

1. Combination 1



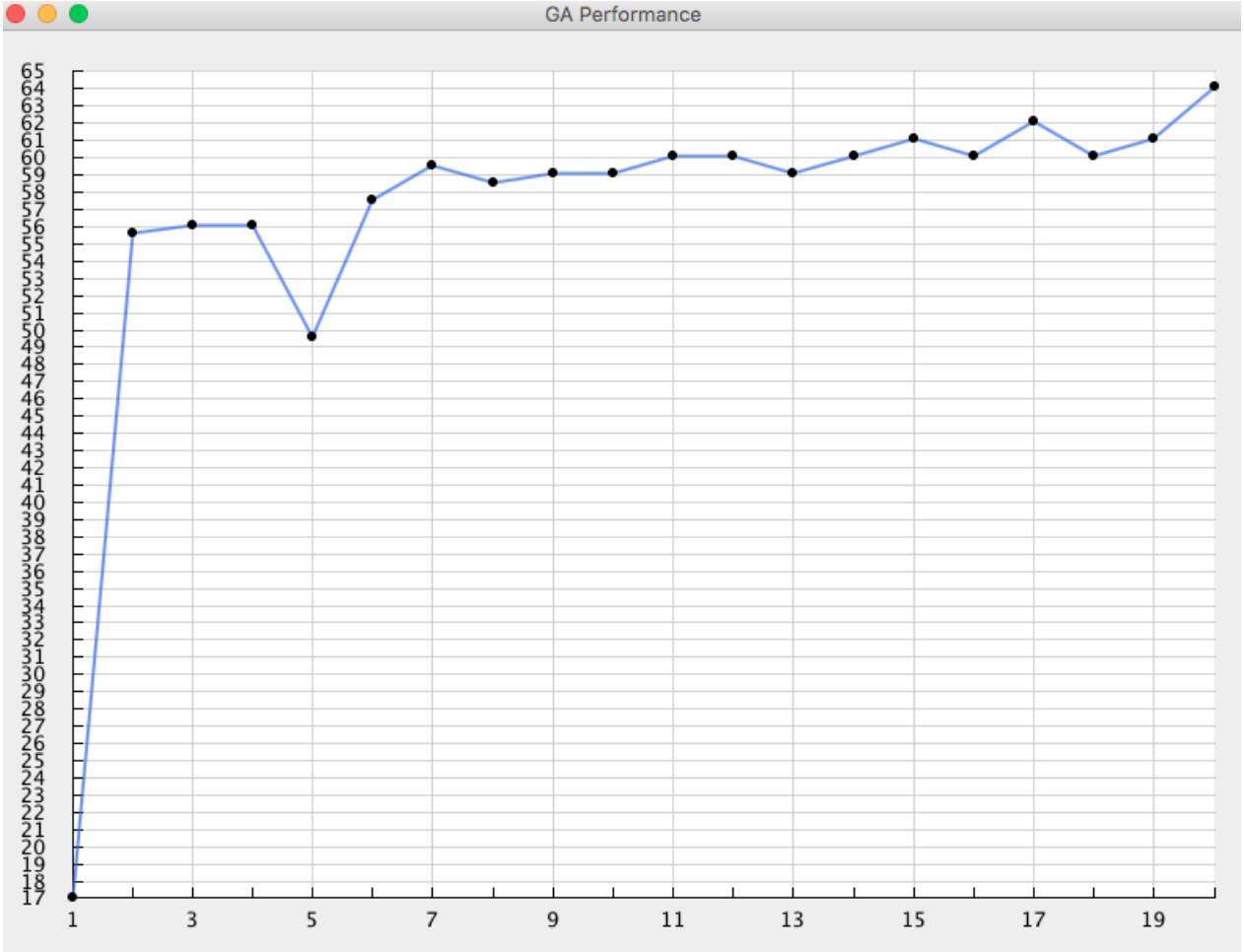
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
1	10	0.95	0.001	2,3	55	48.625

2. Combination 2



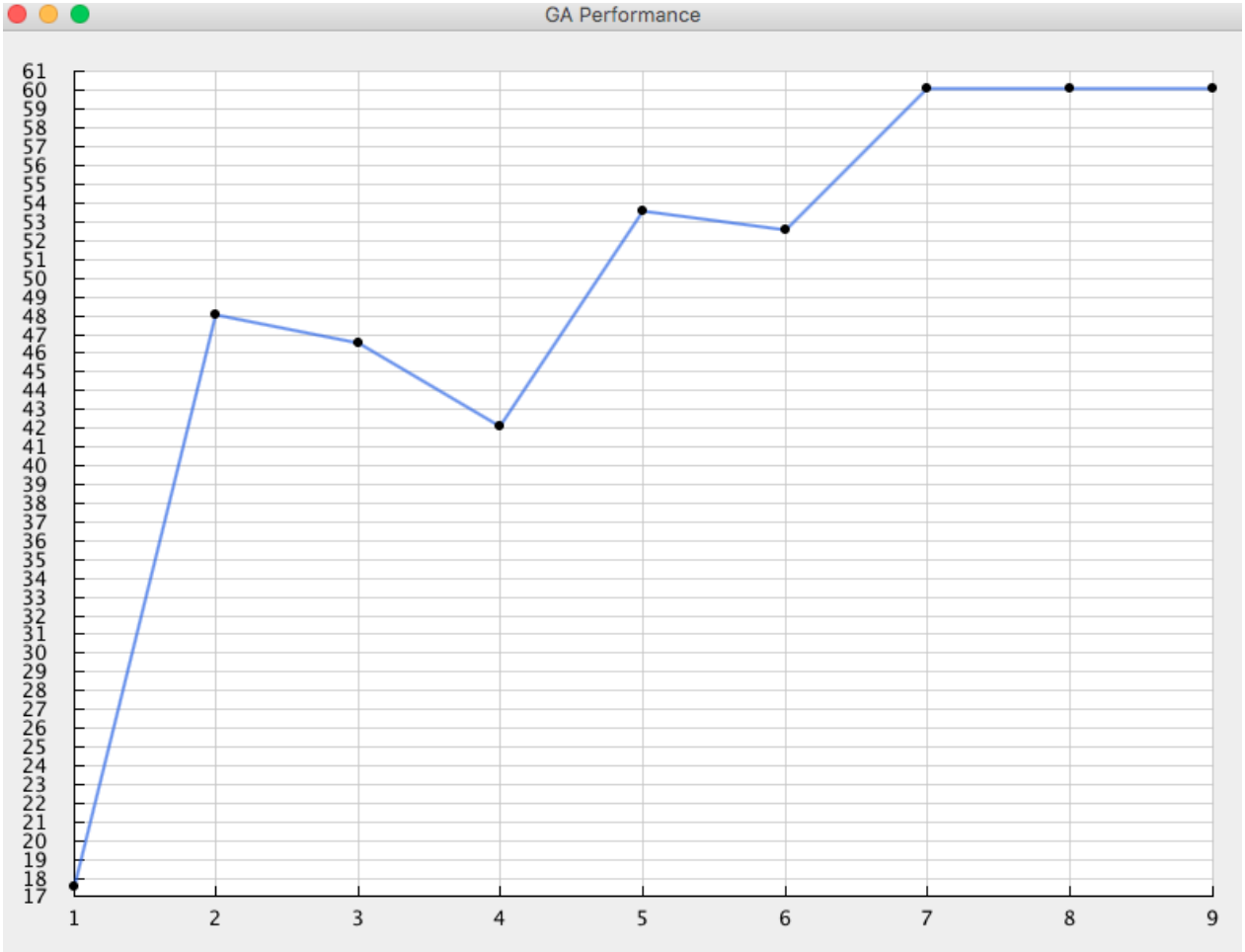
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
2	10	0.90	0.001	2,4	60	46.266666666666666

3. Combination 3



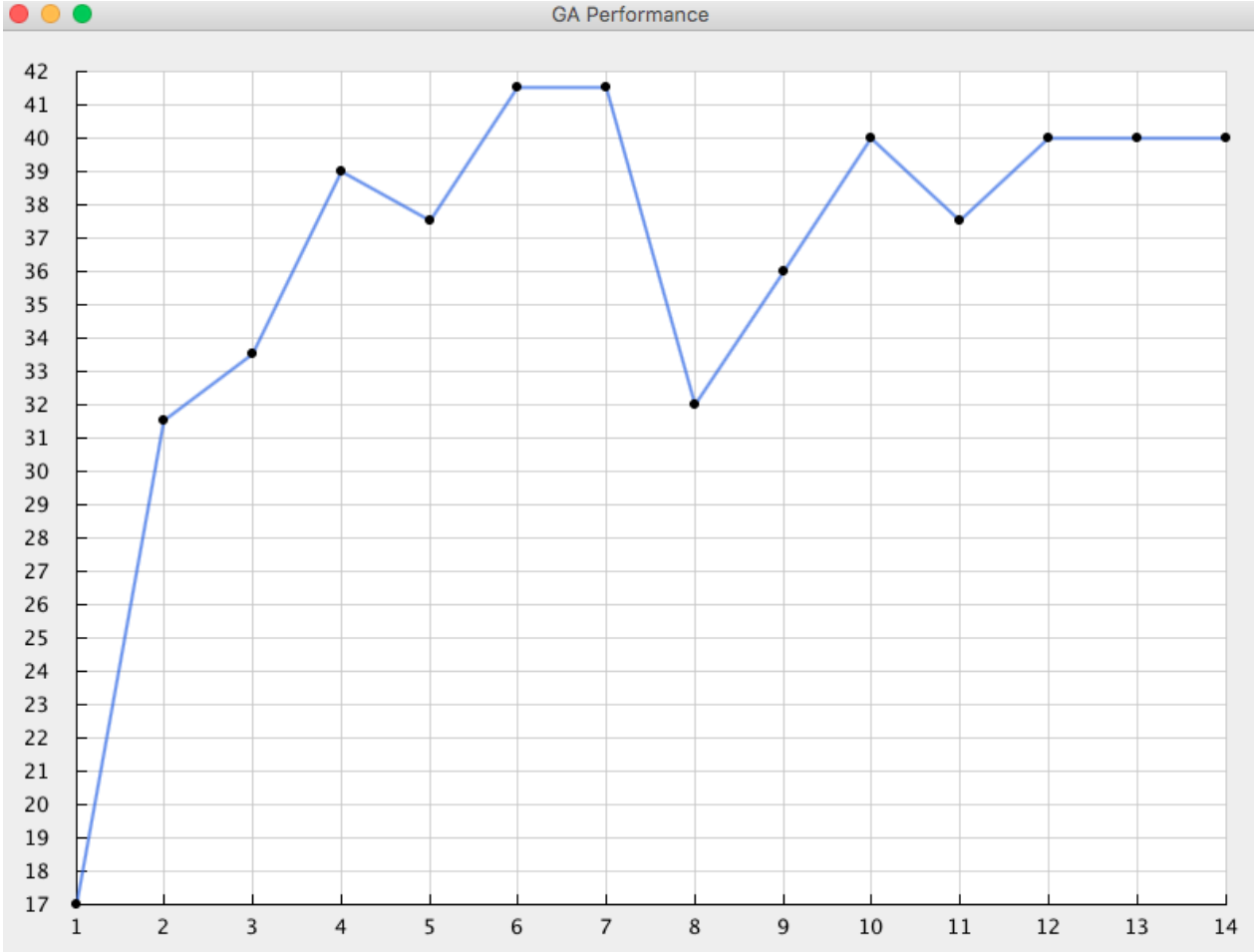
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
3	10	0.85	0.001	1,3	65	56.725

4. Combination 4



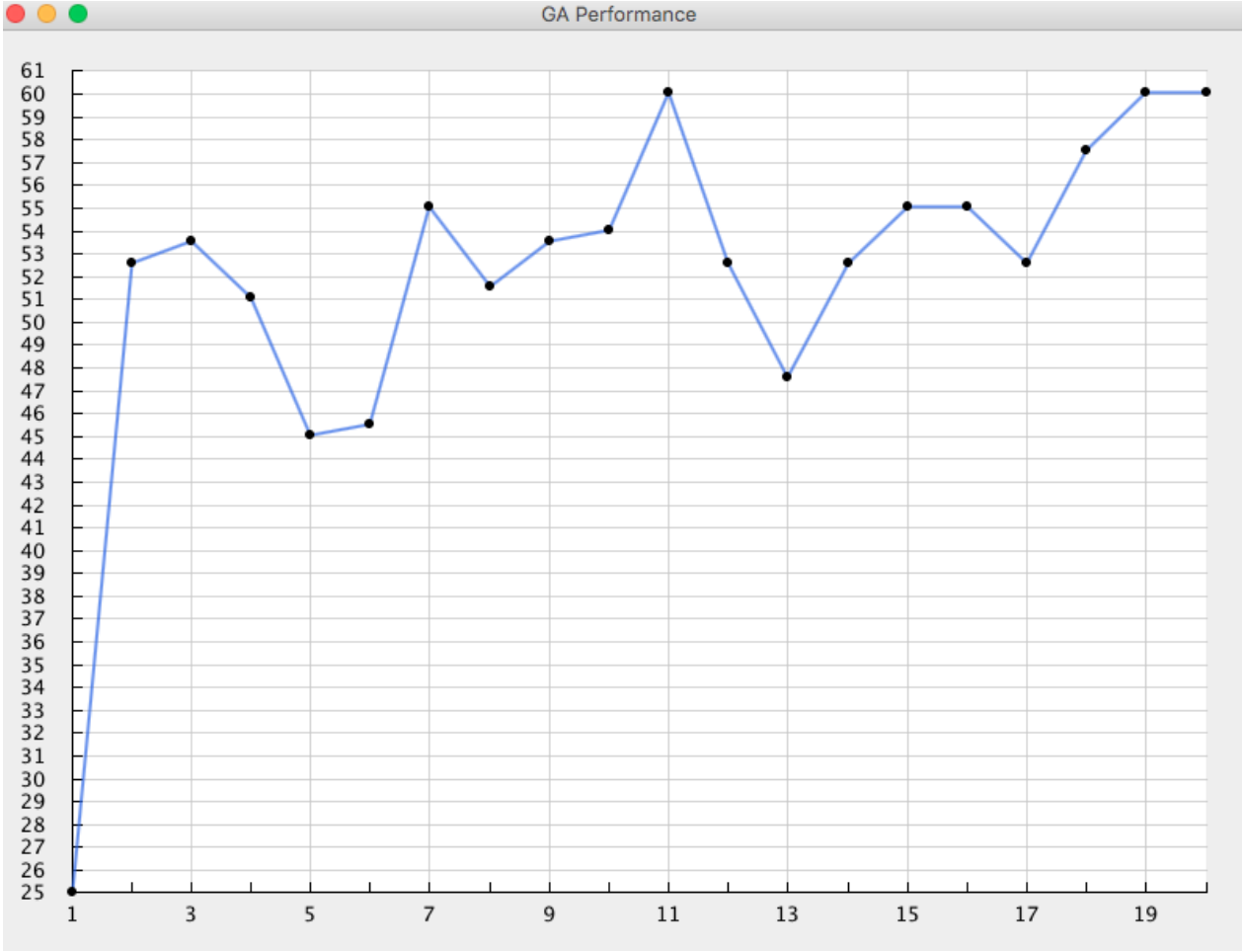
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
4	10	0.95	0.01	2,4	60	48.888888888888886

5. Combination 5



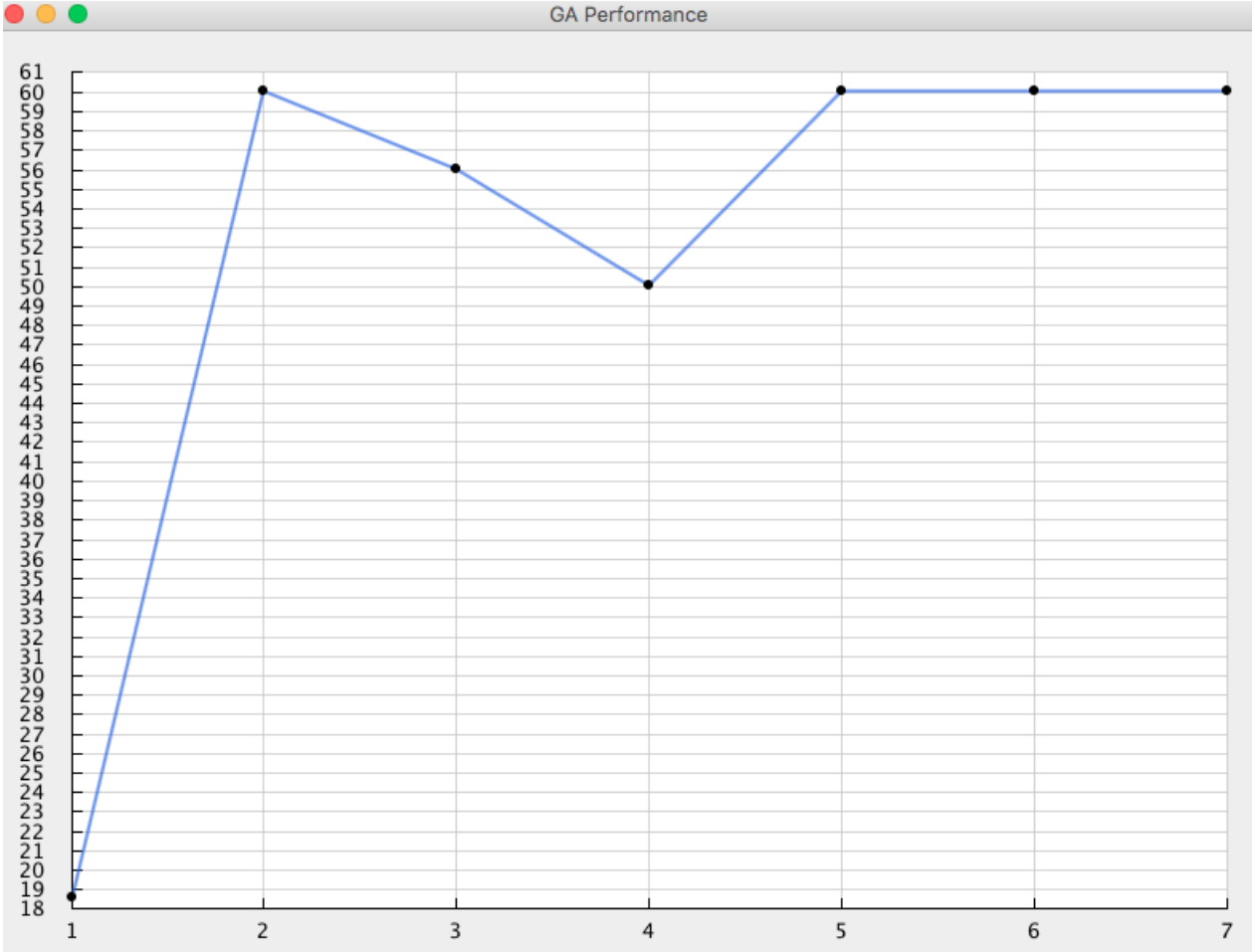
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
5	10	0.90	0.01	2,3	55	36.214285714285715

6. Combination 6



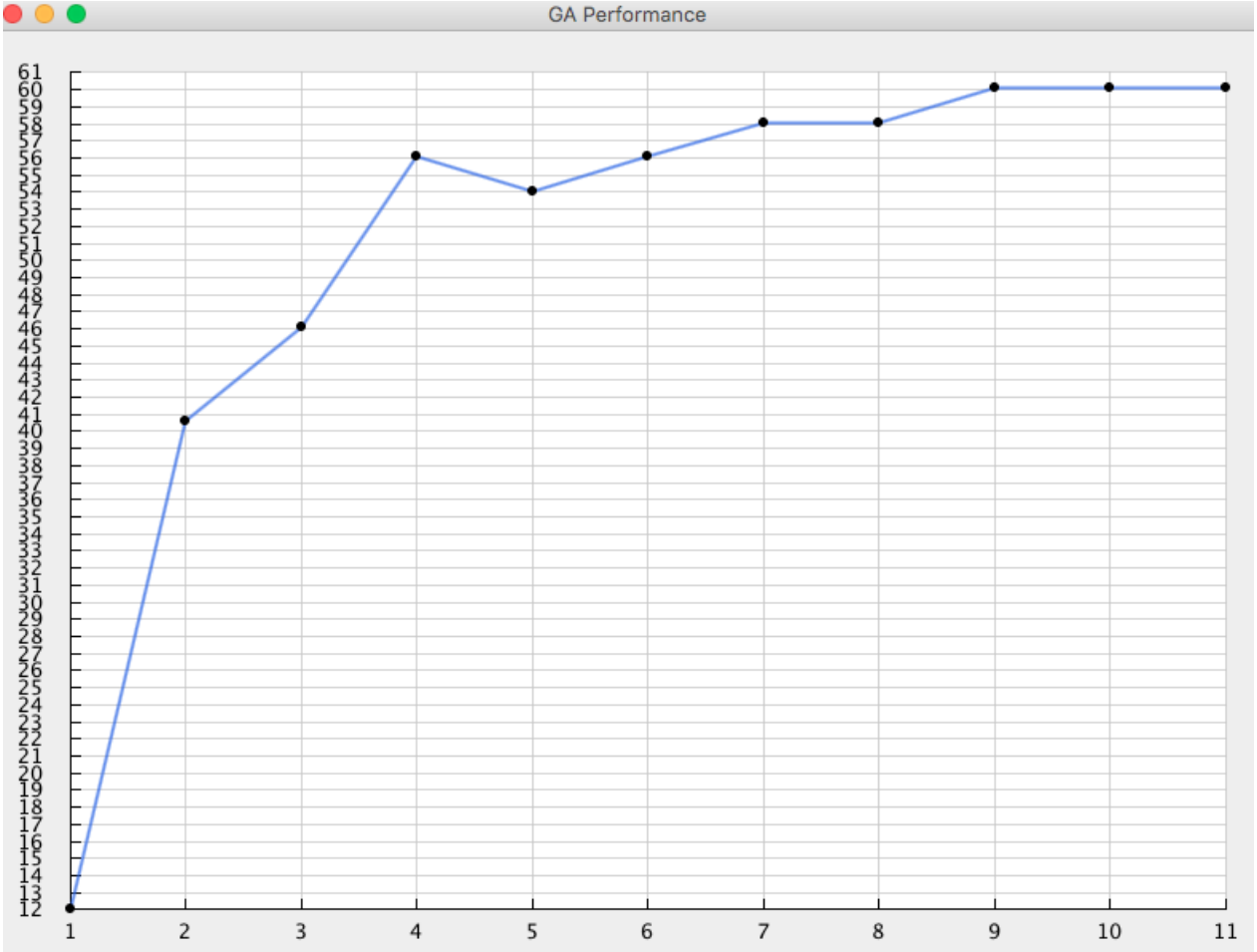
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
6	10	0.85	0.01	3,4,5	60	51.95

7. Combination 7



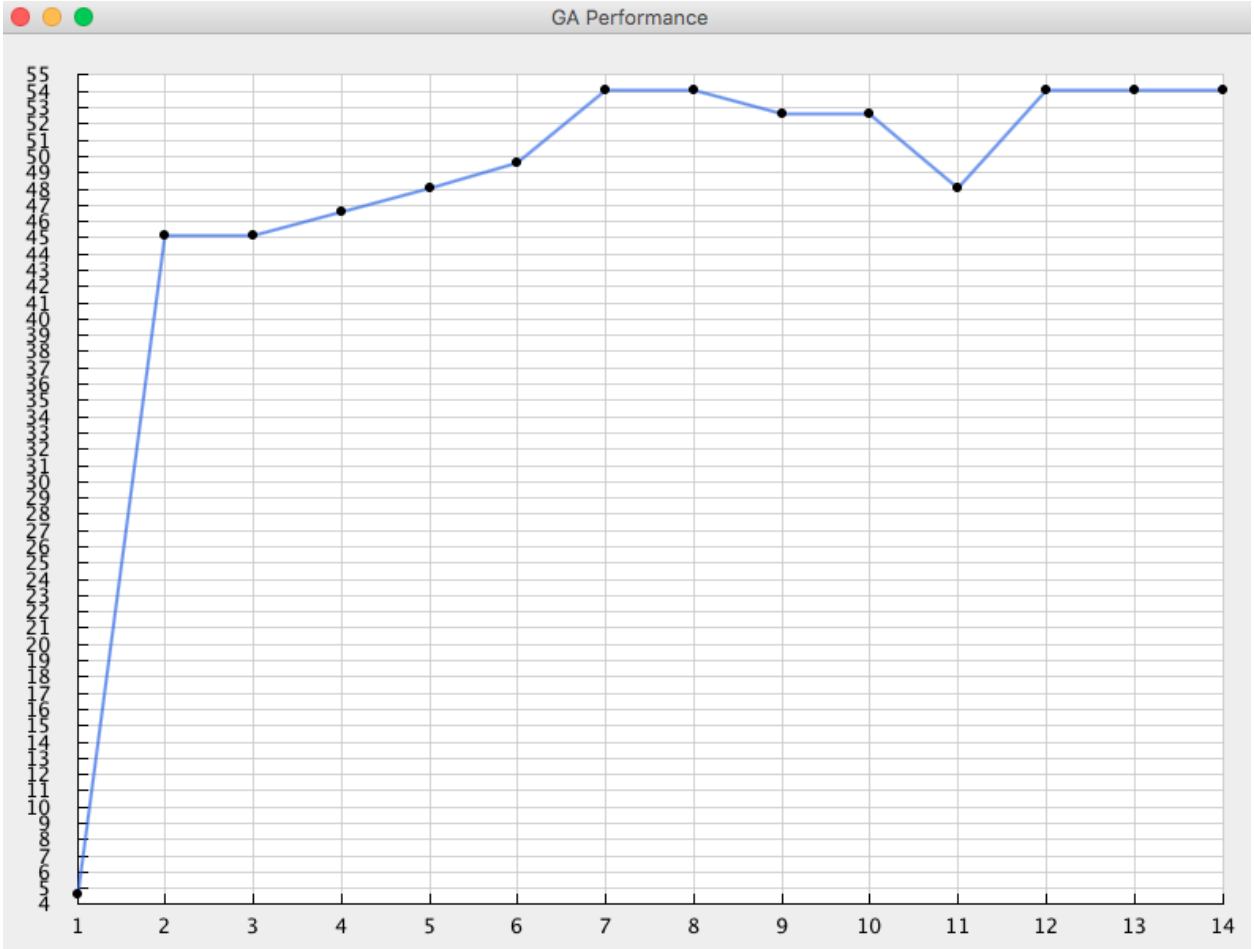
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
7	10	0.95	0.1	2,4	60	52.07142857142857

8. Combination 8



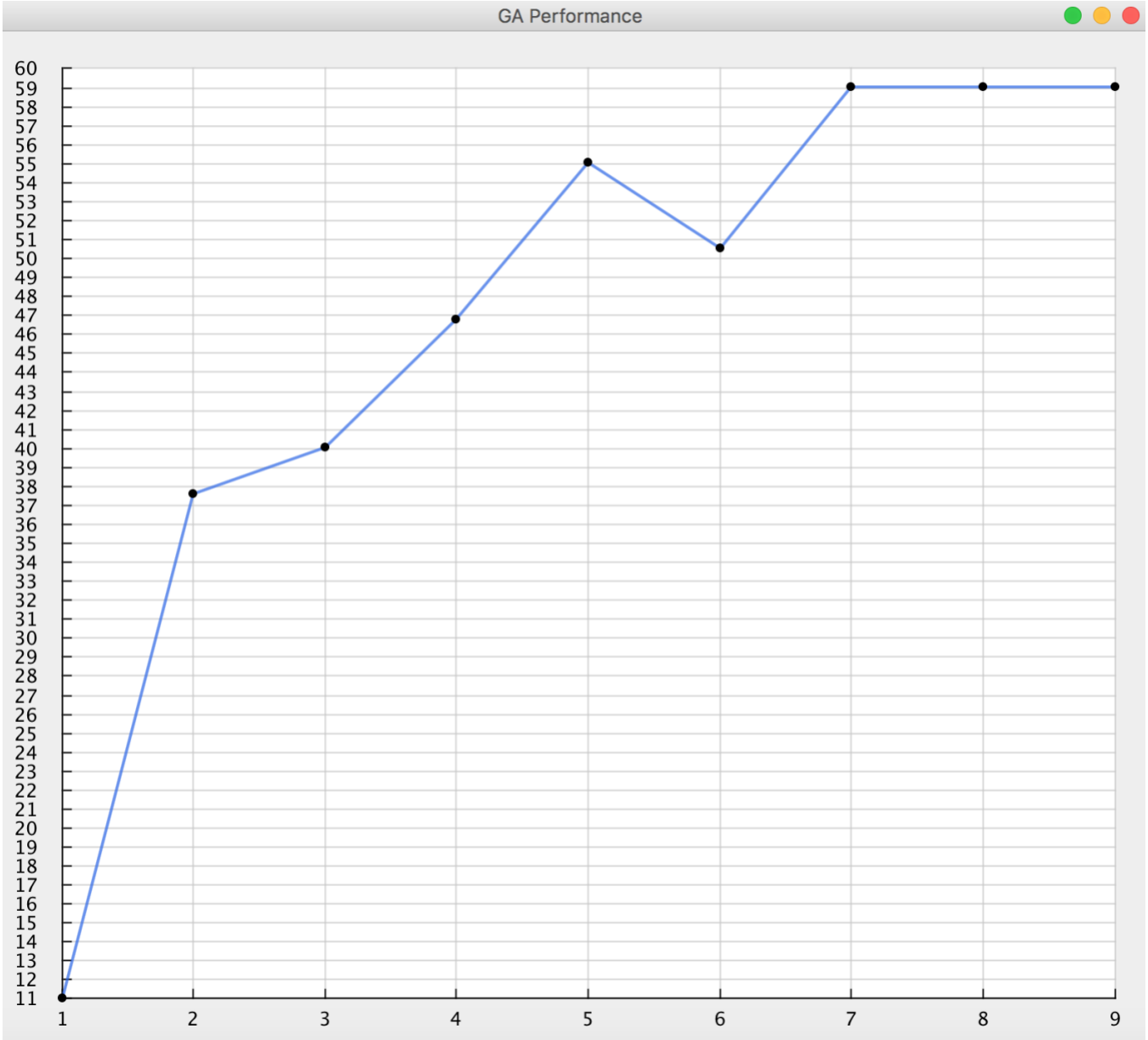
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
8	10	0.9	0.1	2,4	60	50.95454545454545

9. Combination 9



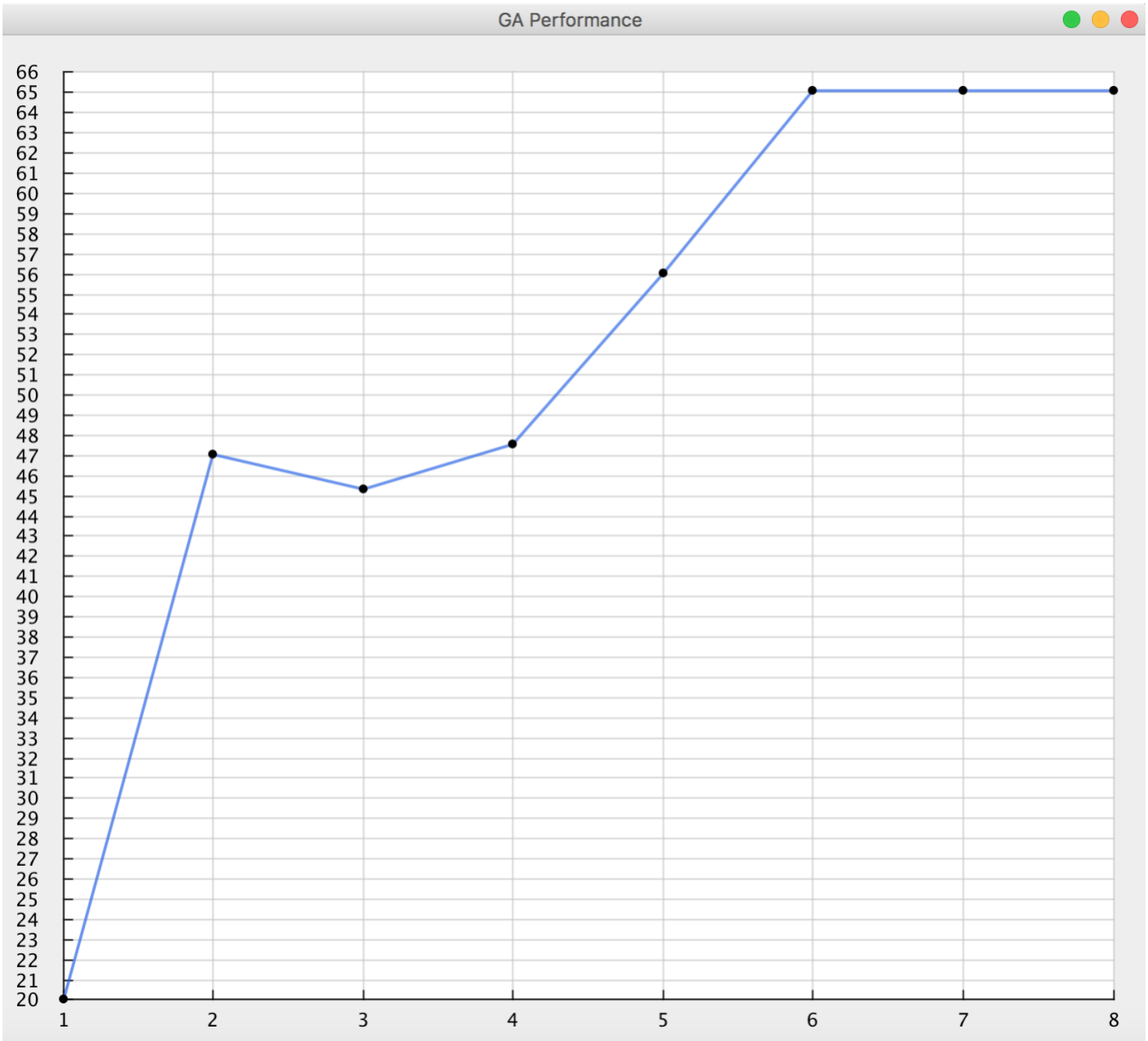
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
9	10	0.85	0.1	3,4,5	60	47.25

10.Combination 10



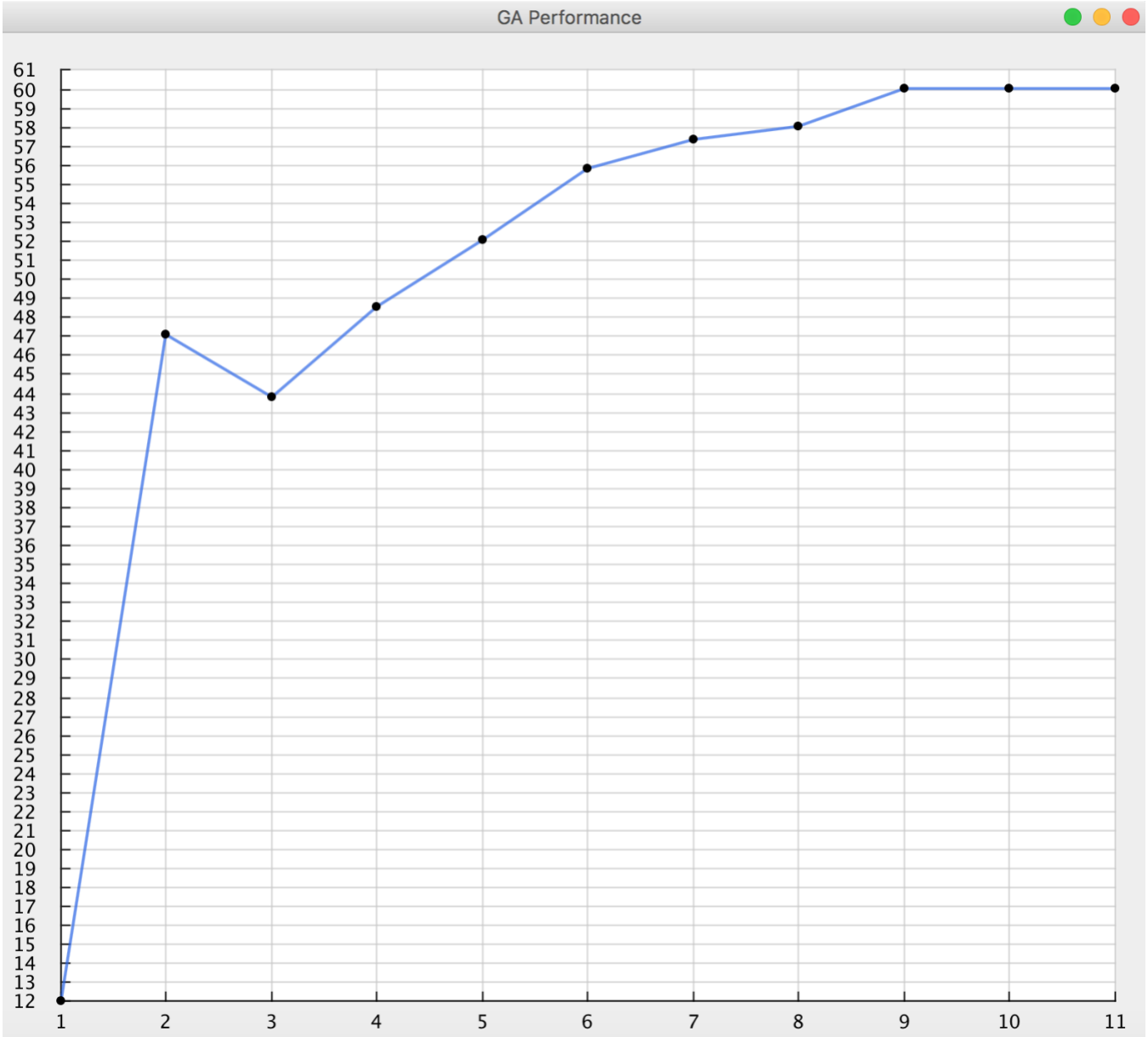
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
10	20	0.95	0.001	1,3	65	46.41666

11.Combination 11



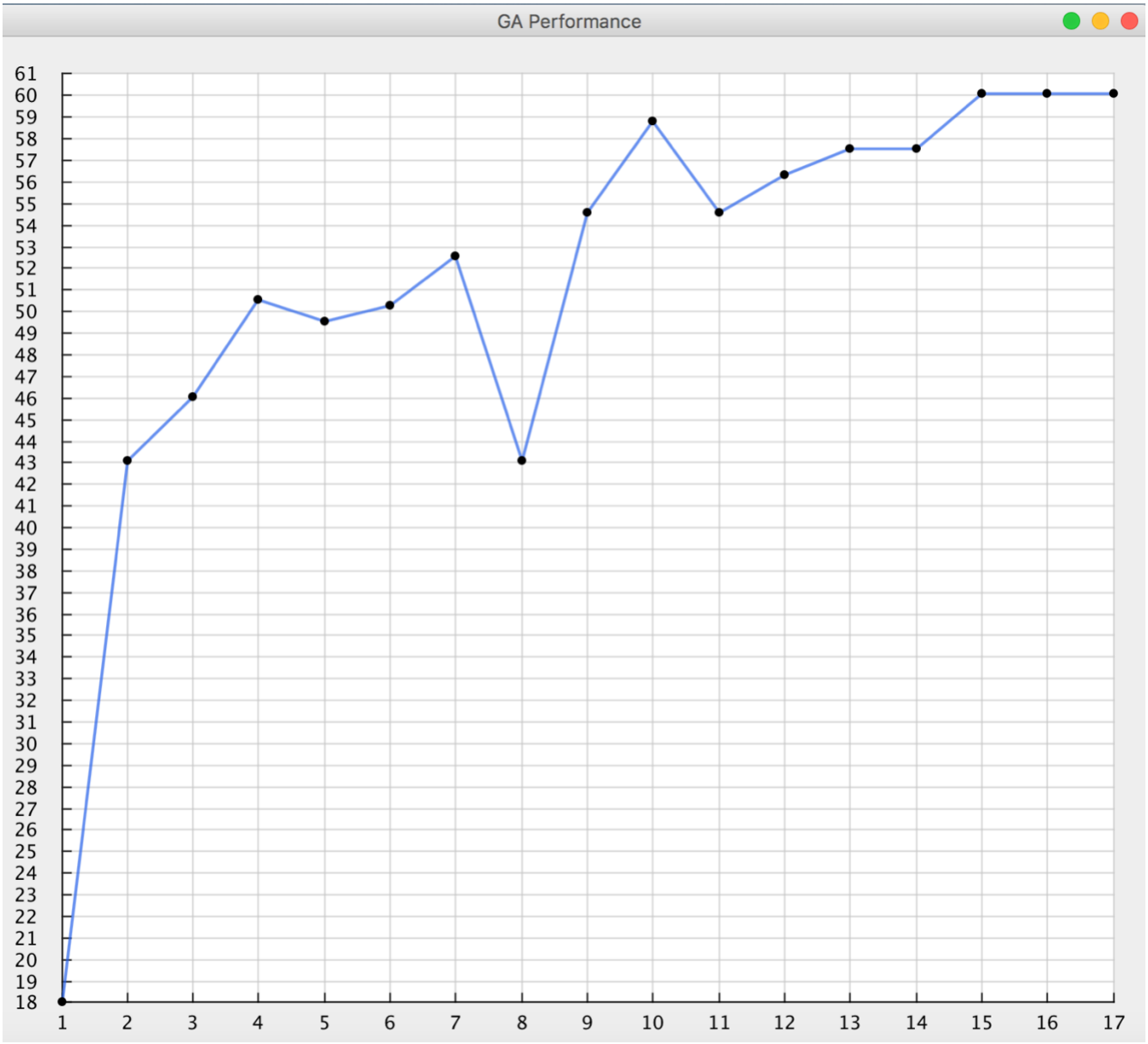
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
11	20	0.90	0.001	1,3	65	51.34375

12.Combination 12



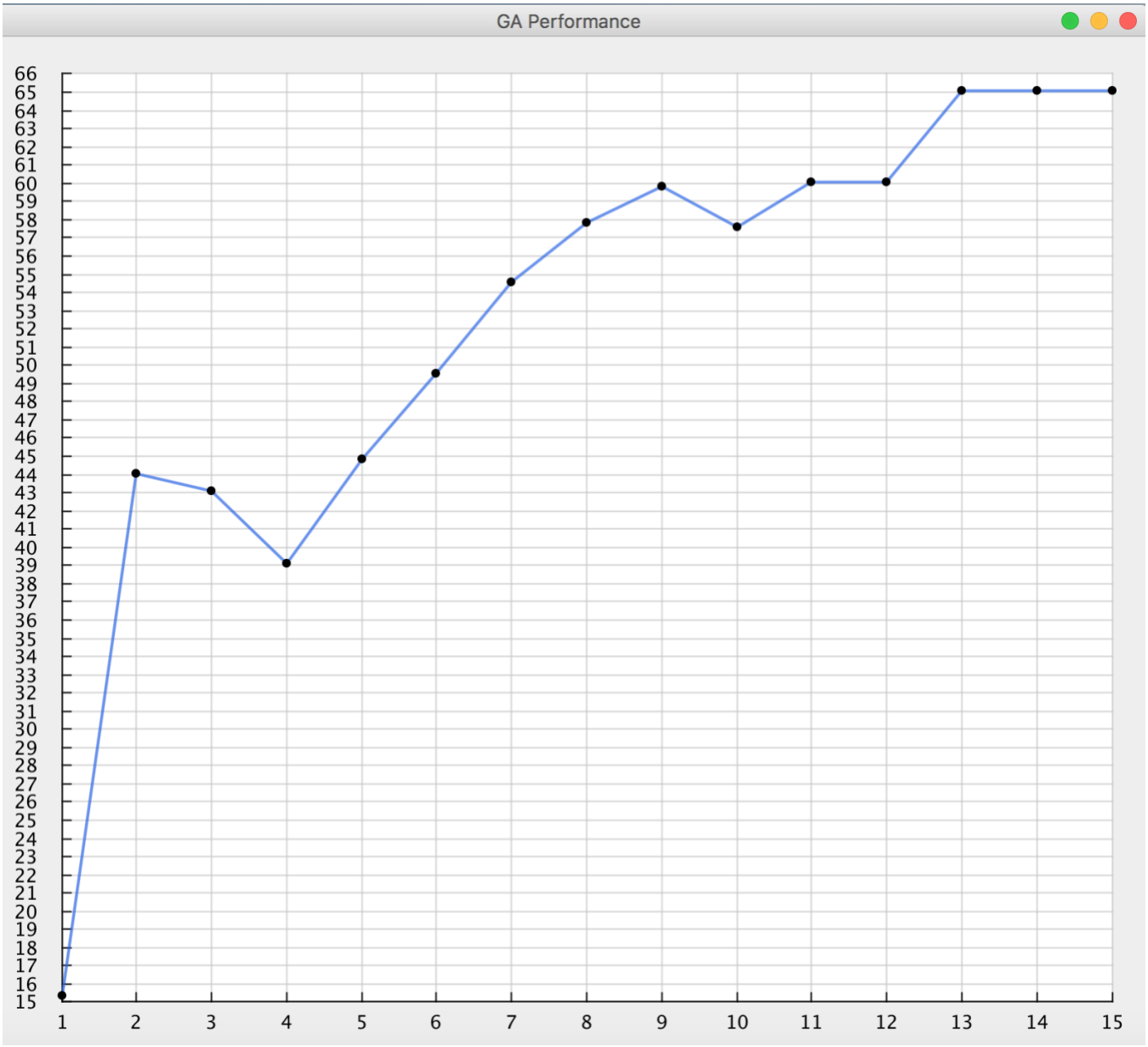
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
12	20	0.85	0.001	3,4,5	60	50.386

13.Combination 13



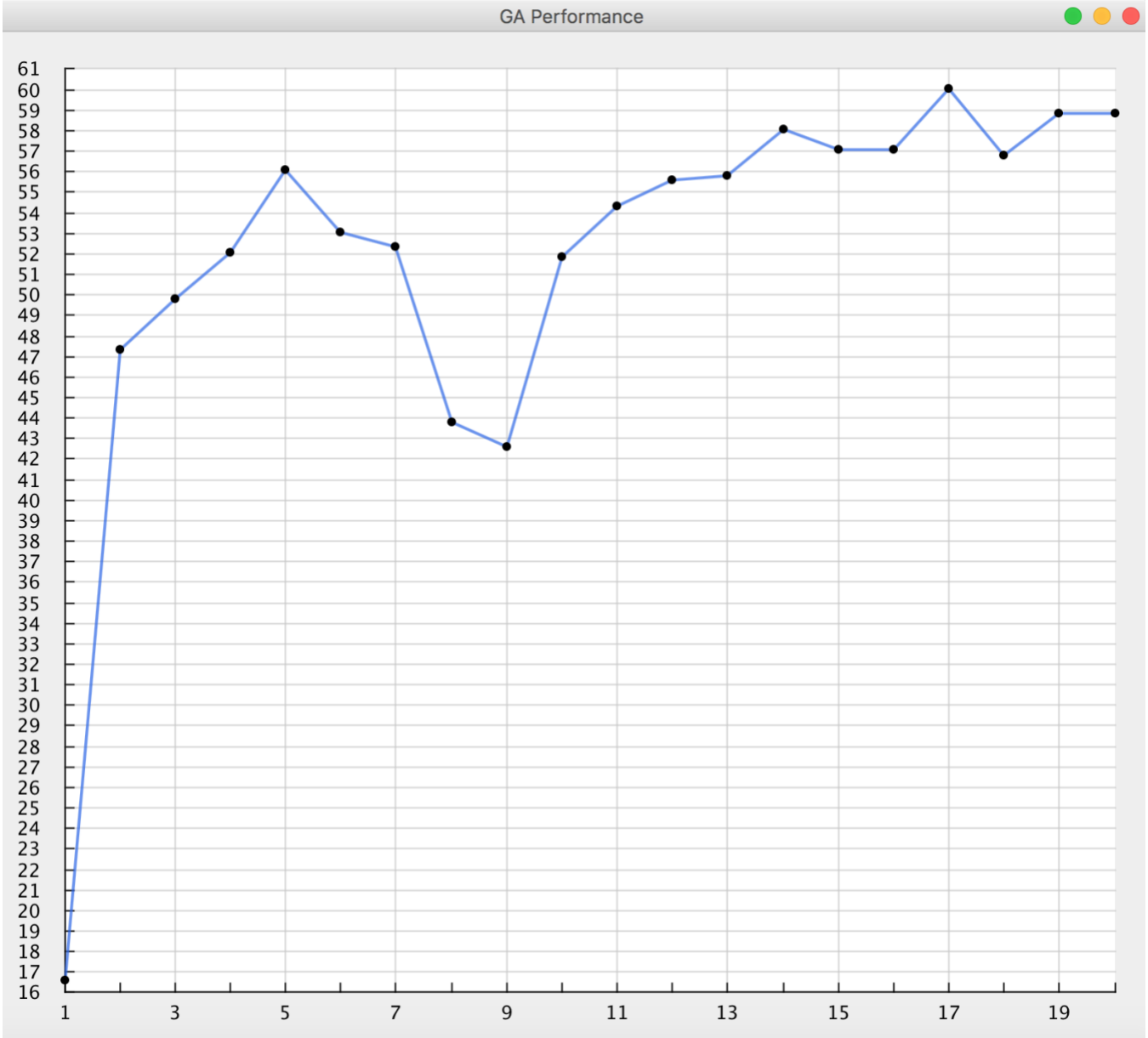
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
13	20	0.95	0.01	2,4	60	51.279411764705884

14.Combination 14



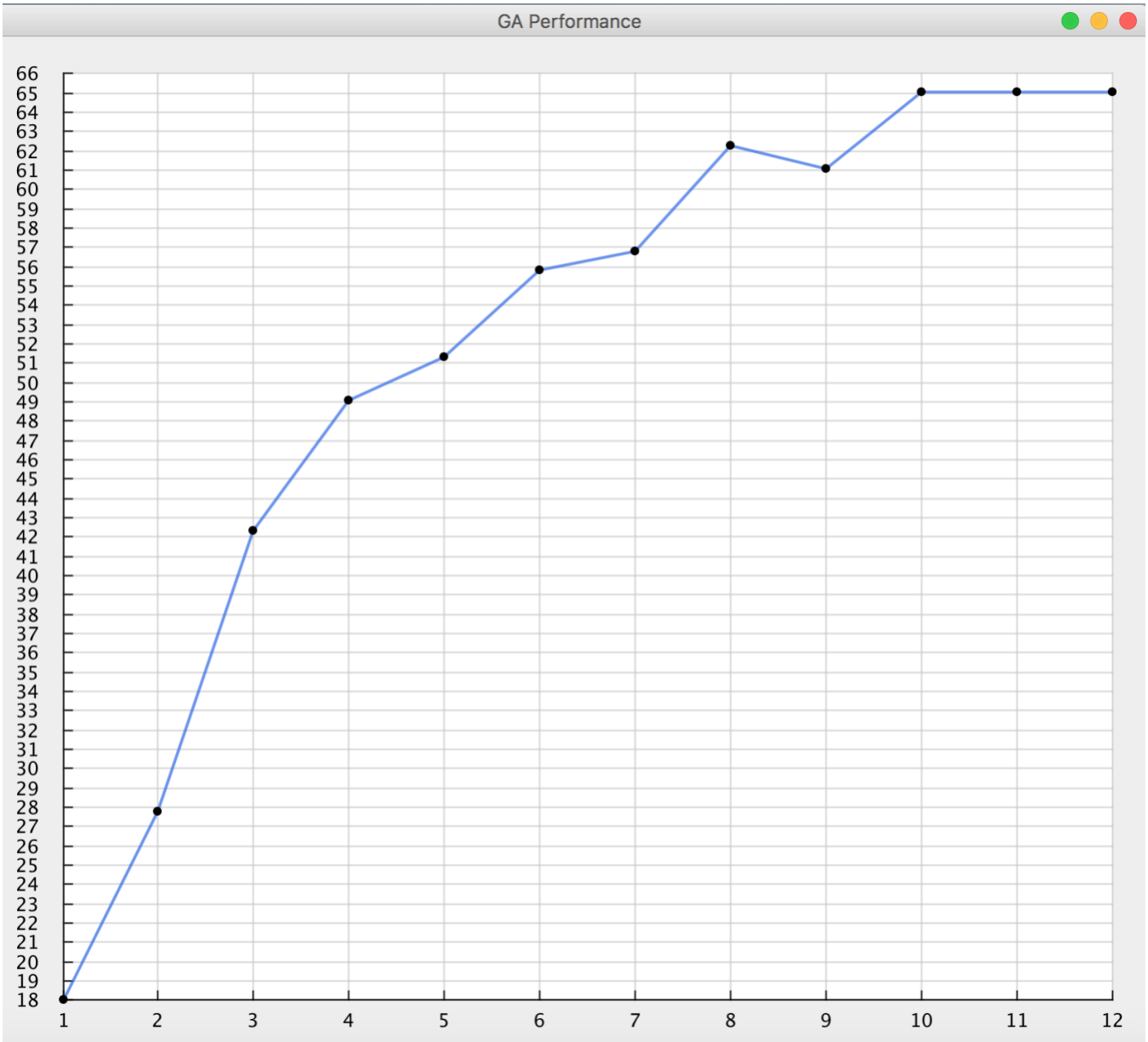
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
14	20	0.90	0.01	1,3	65	52.0

15.Combination 15



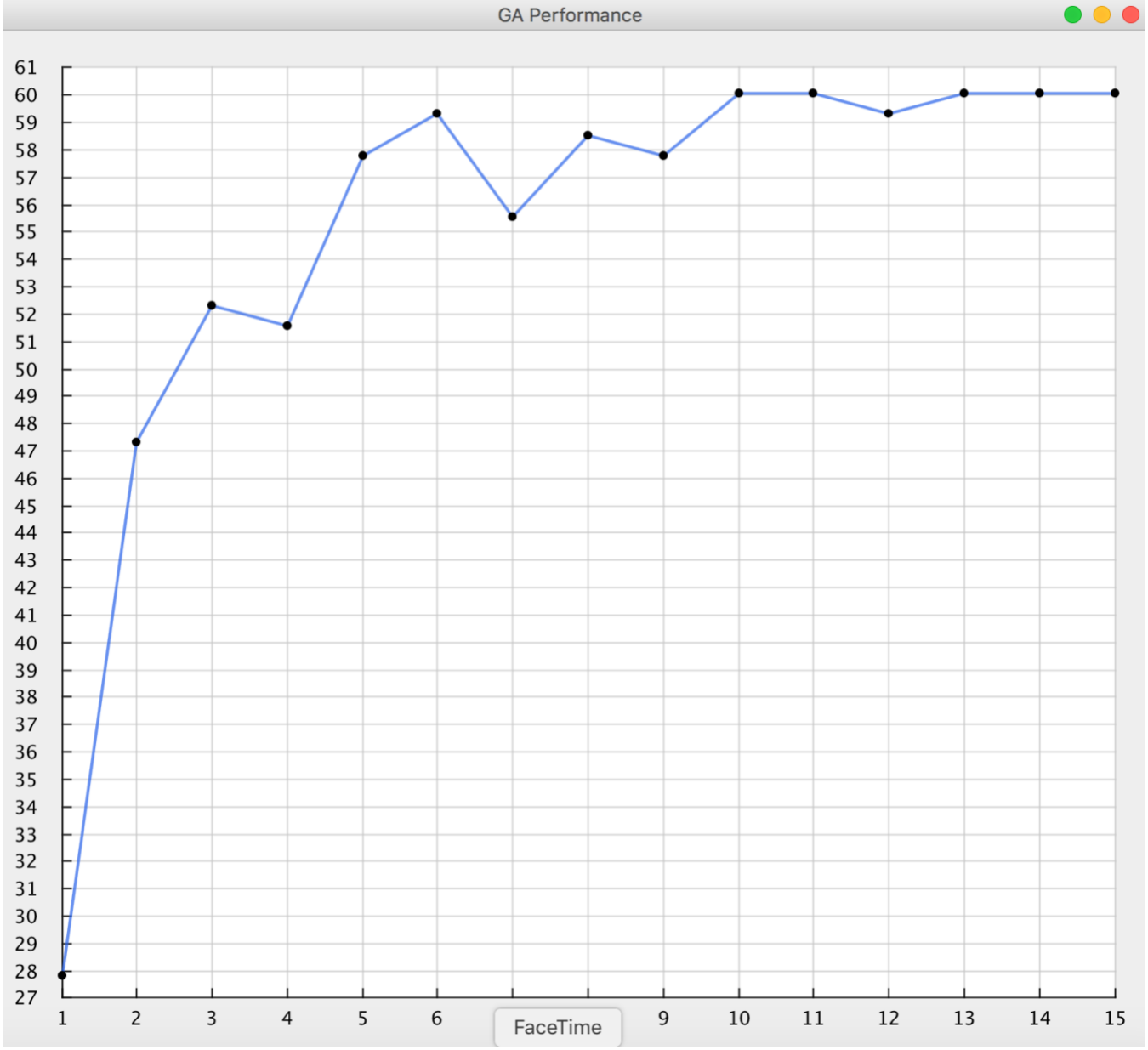
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
15	20	0.85	0.01	3,4,5	60	51.825

16.Combination 16



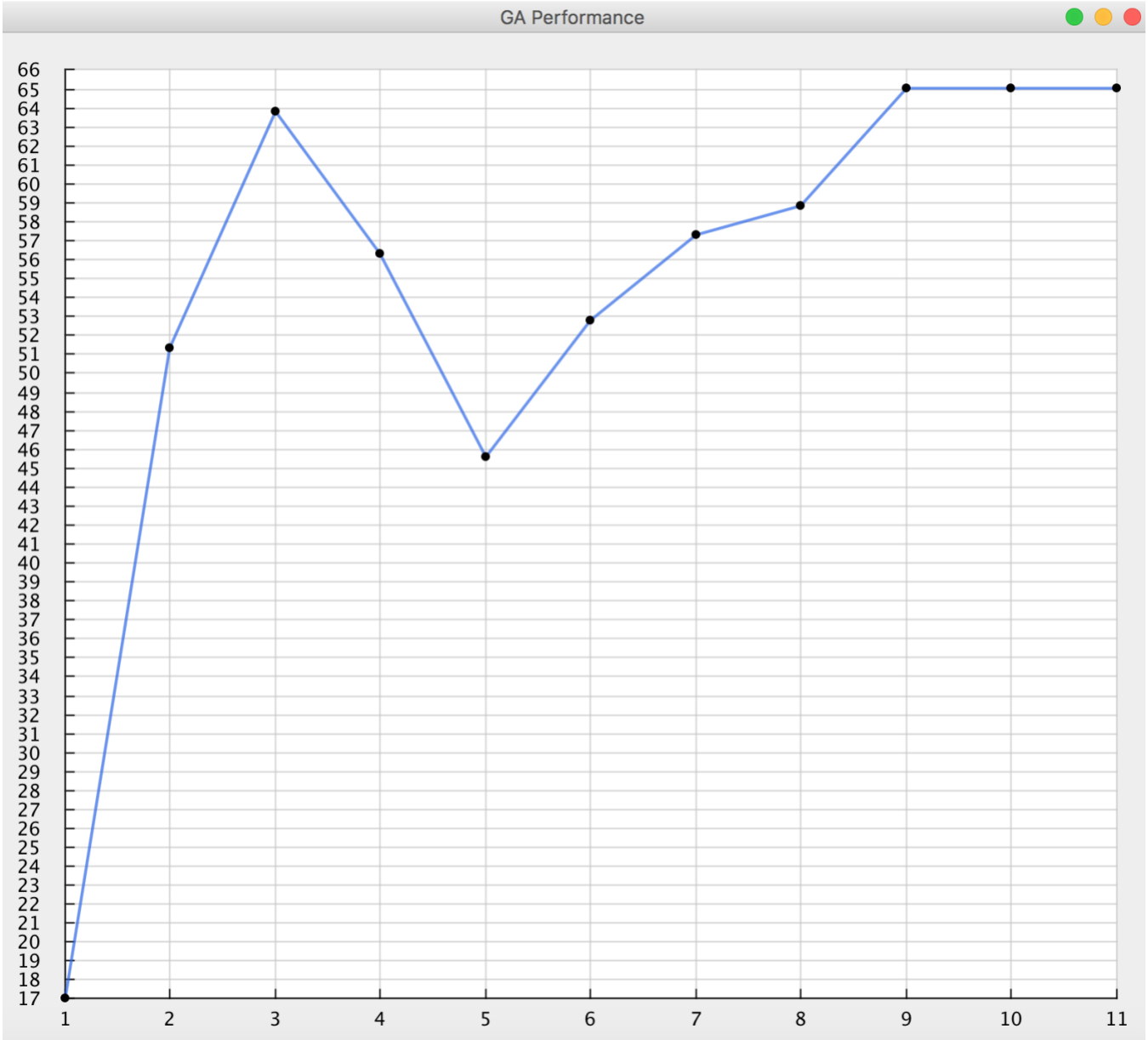
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
16	20	0.95	0.1	1,3	65	51.583333333333336

17.Combination 17



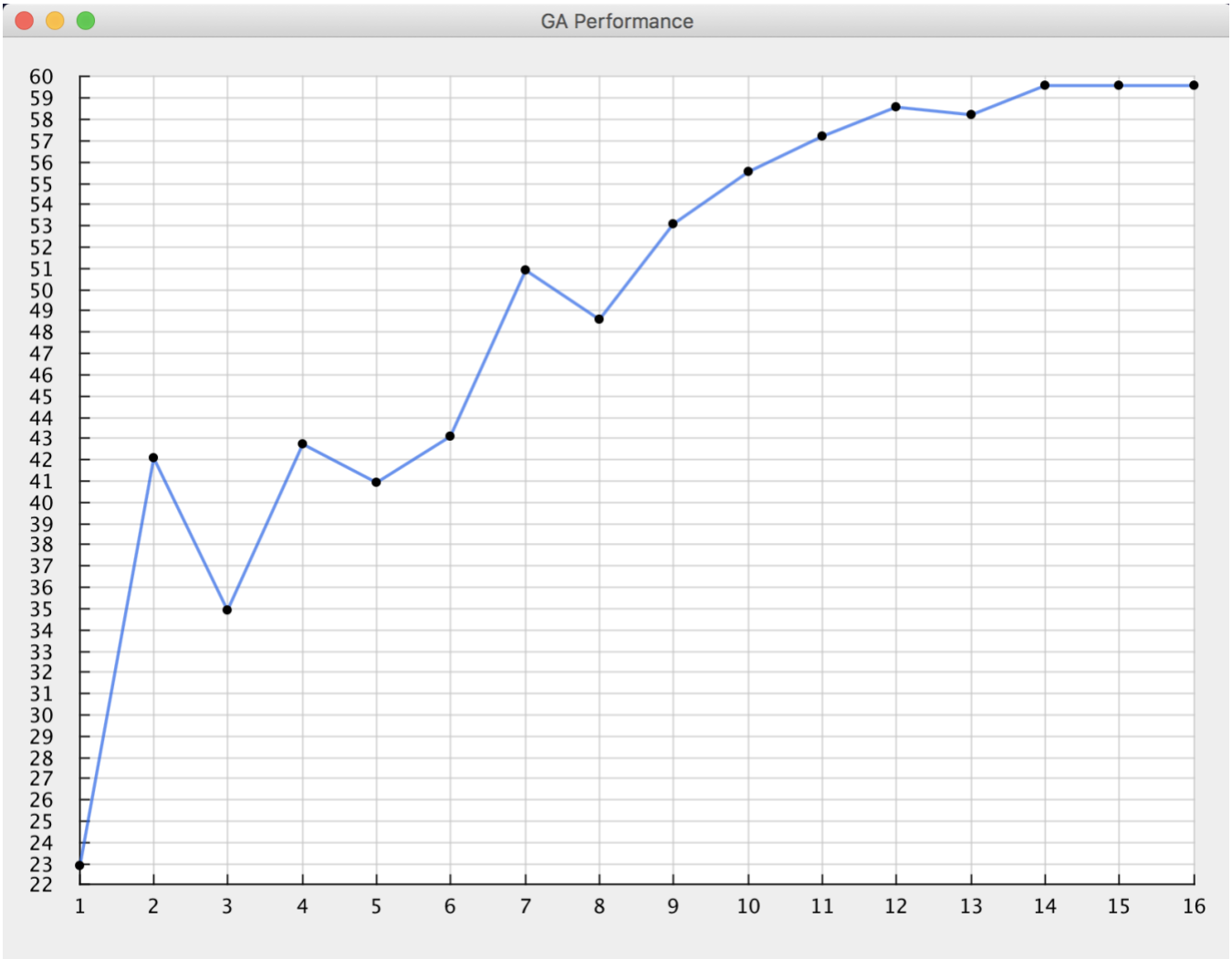
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
17	20	0.90	0.1	3,4,5	60	55.11666666666667

18.Combination 18



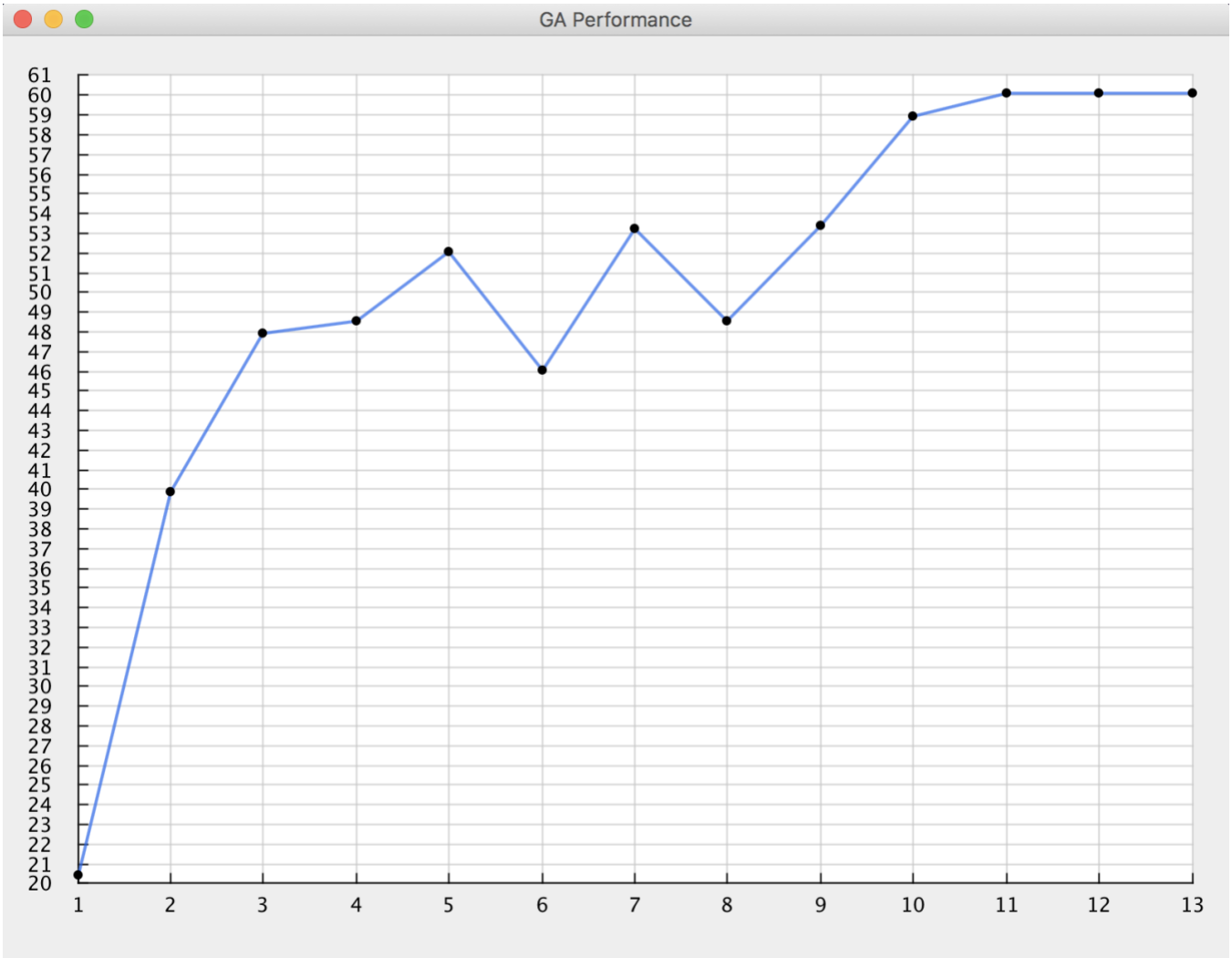
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
18	20	0.85	0.1	1,3	65	54.31818181818182

19.Combination 19



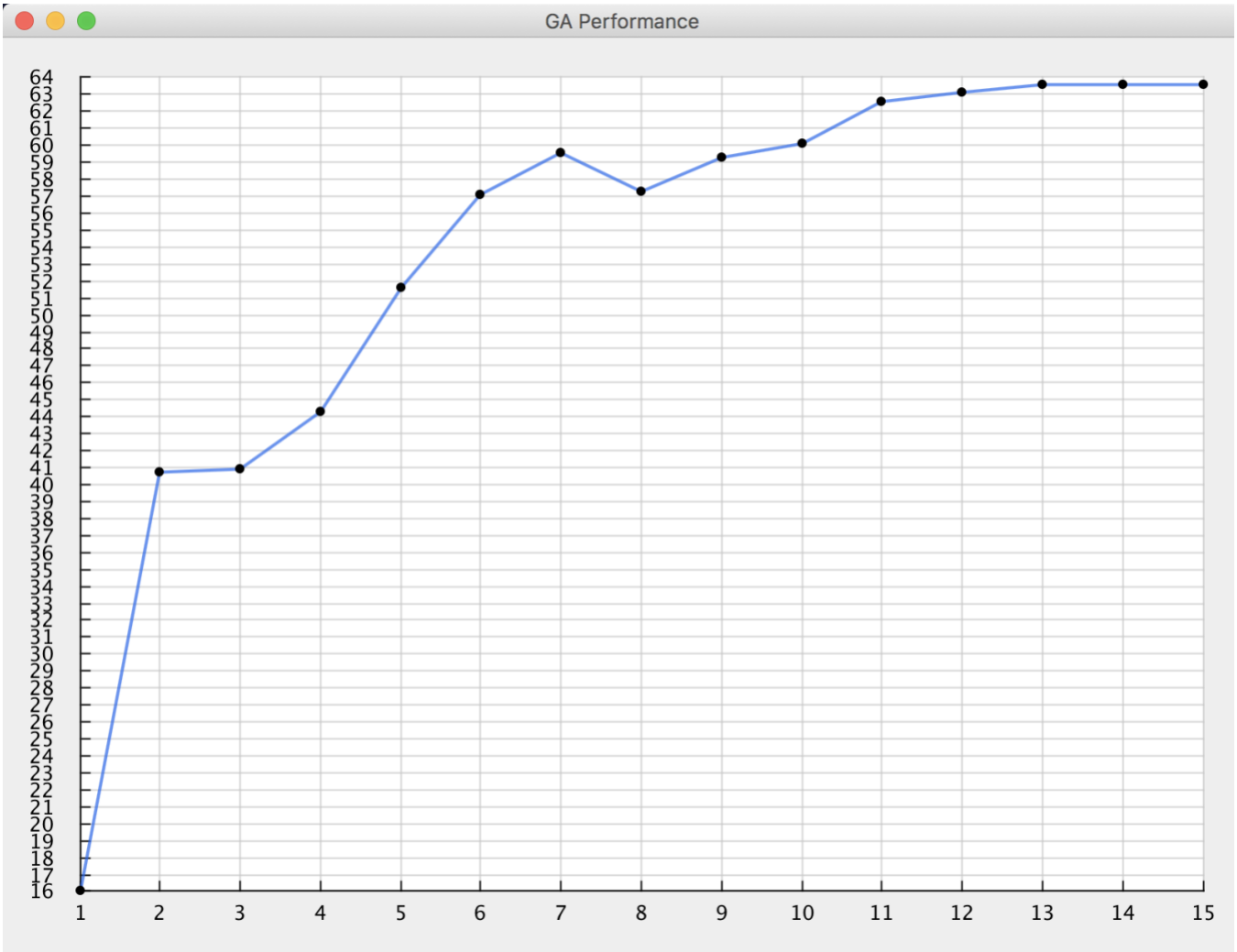
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
19	30	0.95	0.001	1,3	65	49.145833333333336

20.Combination 20



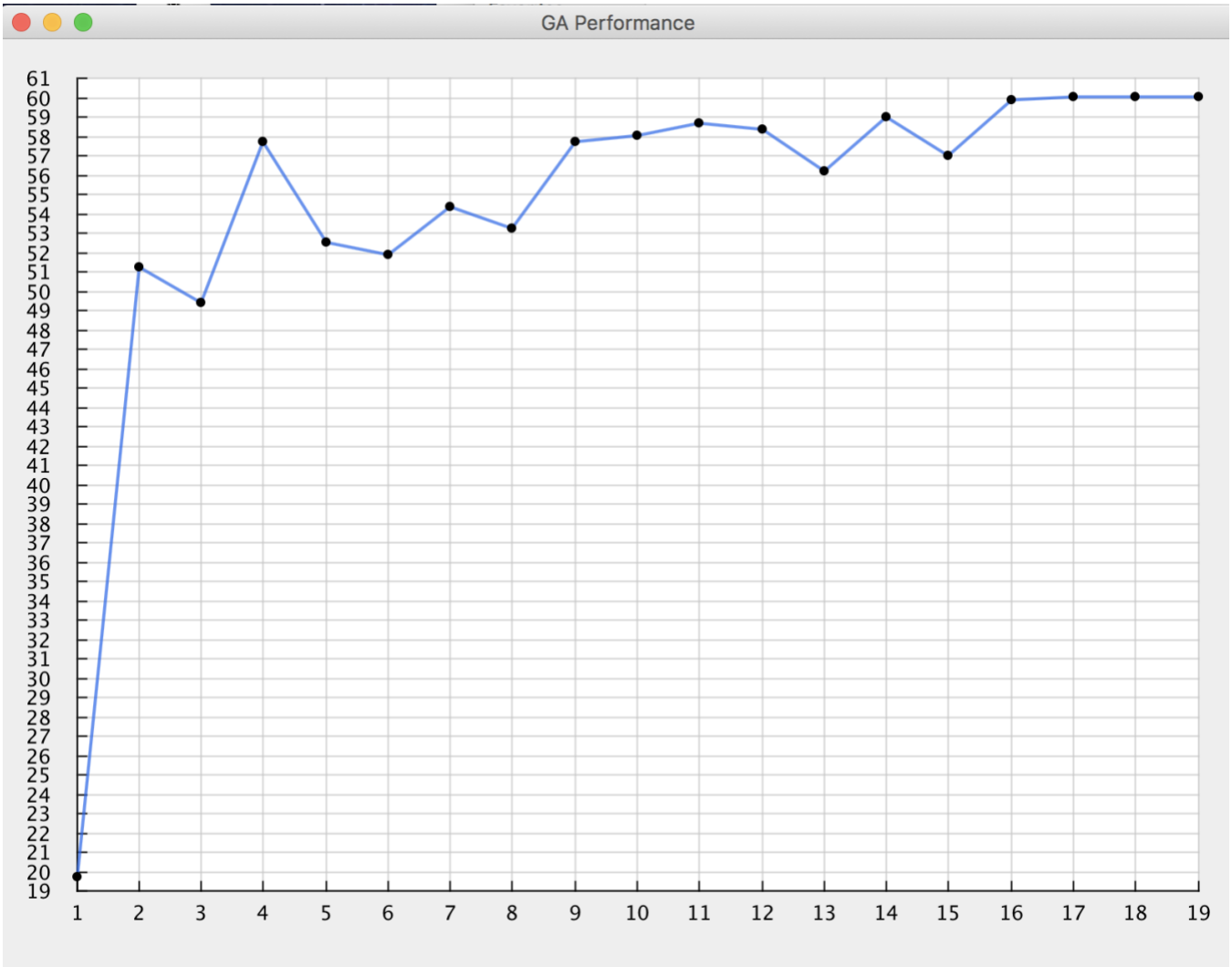
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
20	30	0.90	0.001	2,4	60	49.87179487179487

21.Combination 21



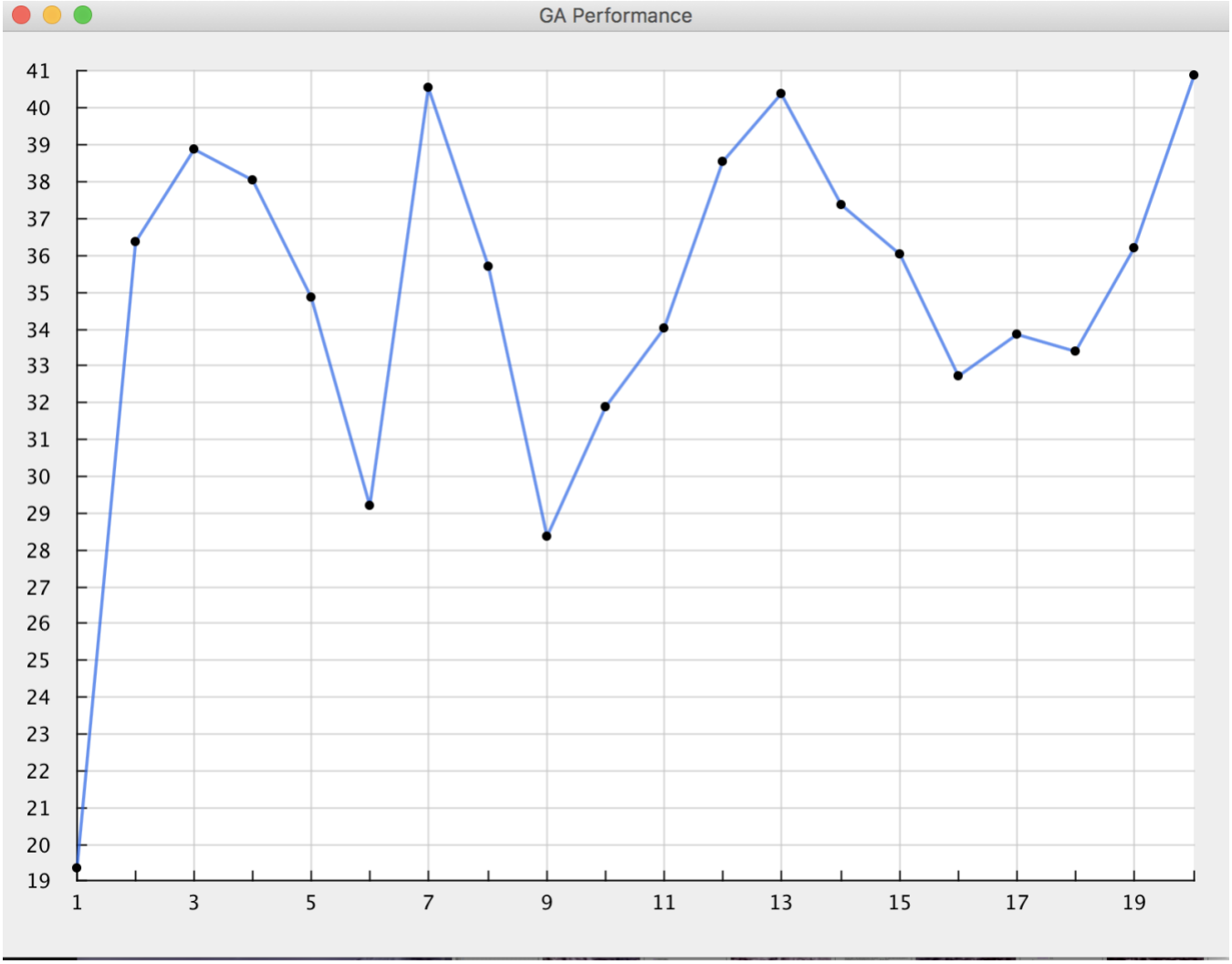
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
21	30	0.85	0.001	1,3	65	53.46666666666667

22.Combination 22



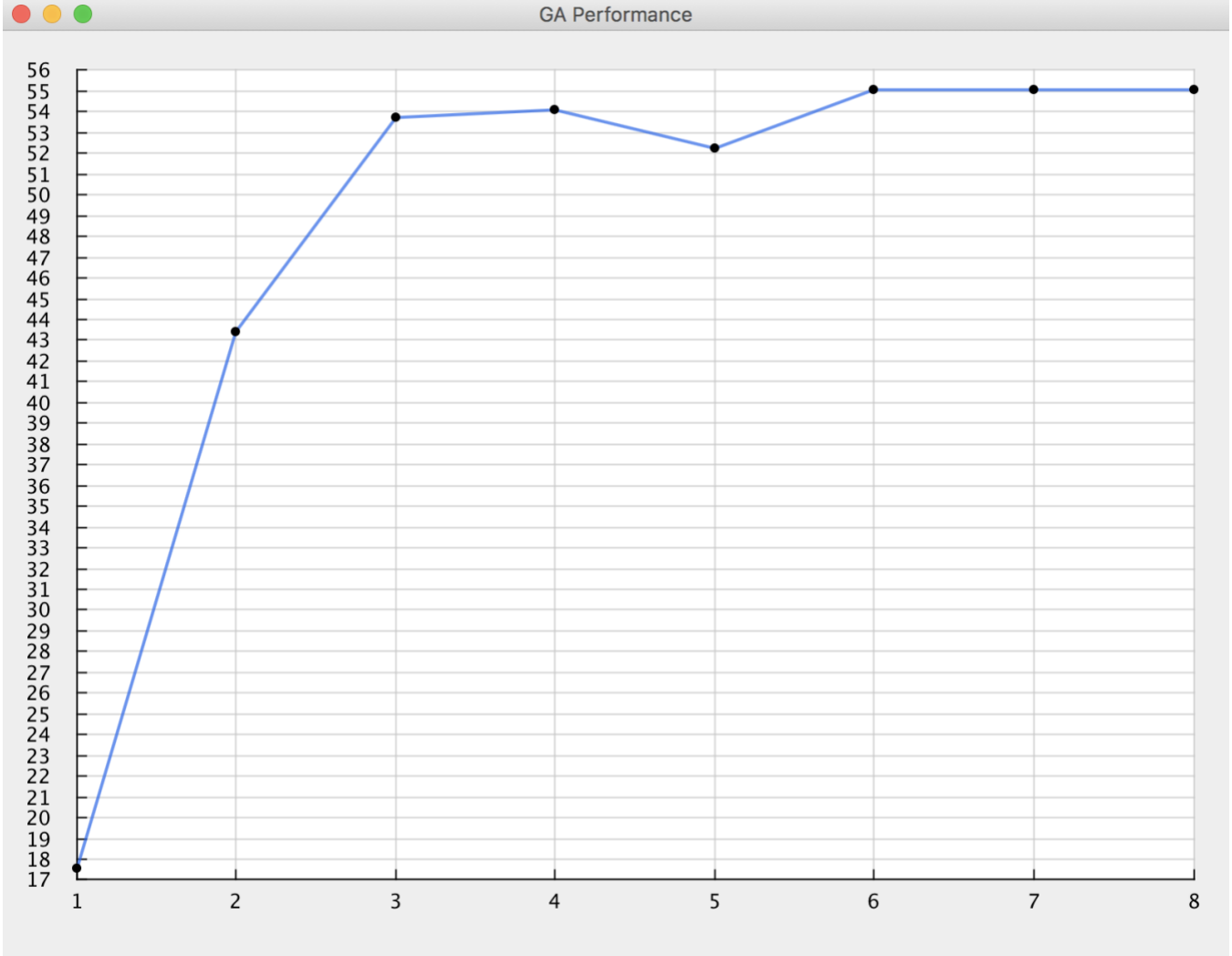
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
22	30	0.95	0.01	1,3	65	54.438596491228076

23.Combination 23



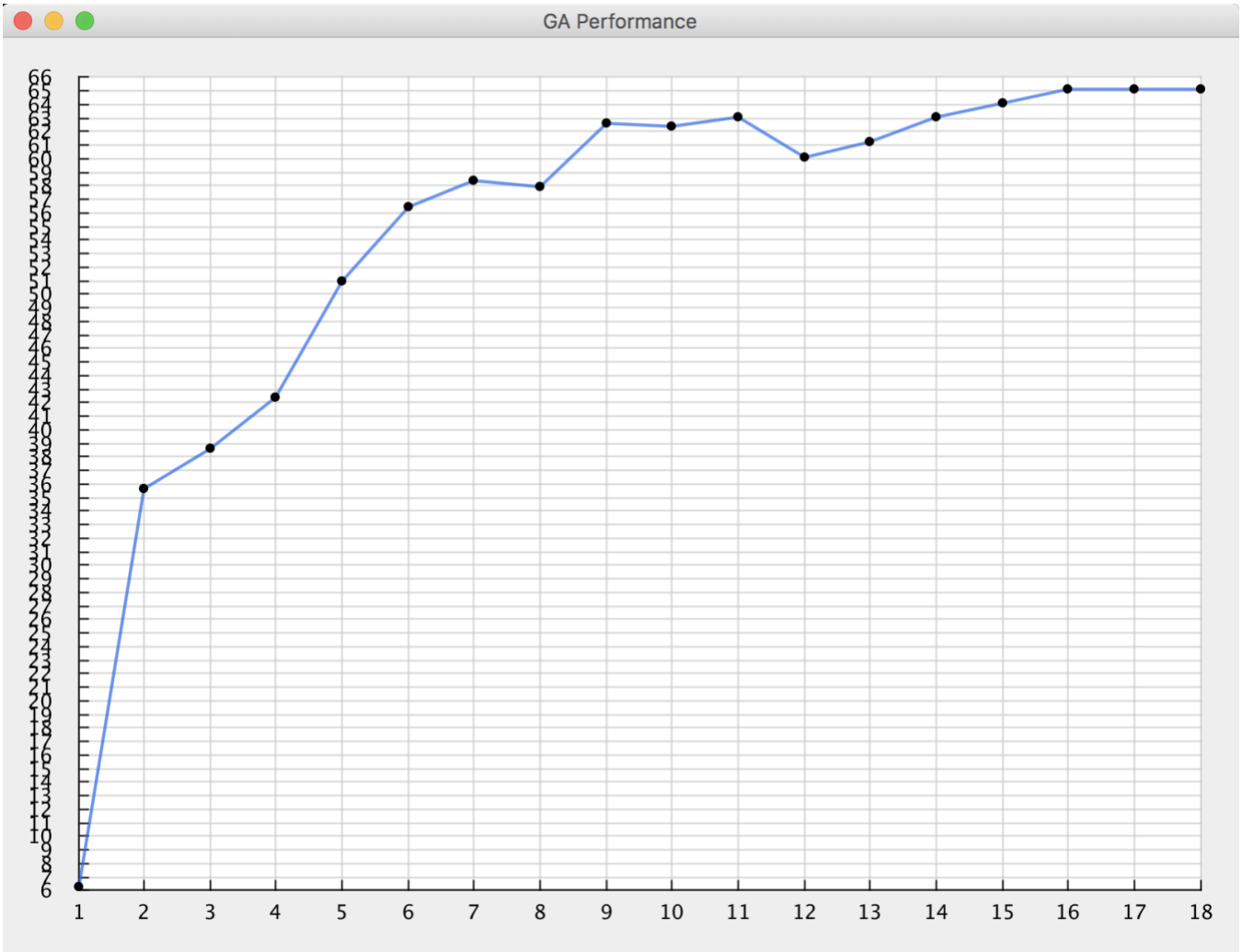
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
23	30	0.90	0.01	1,3	65	34.79166666666667

24.Combination 24



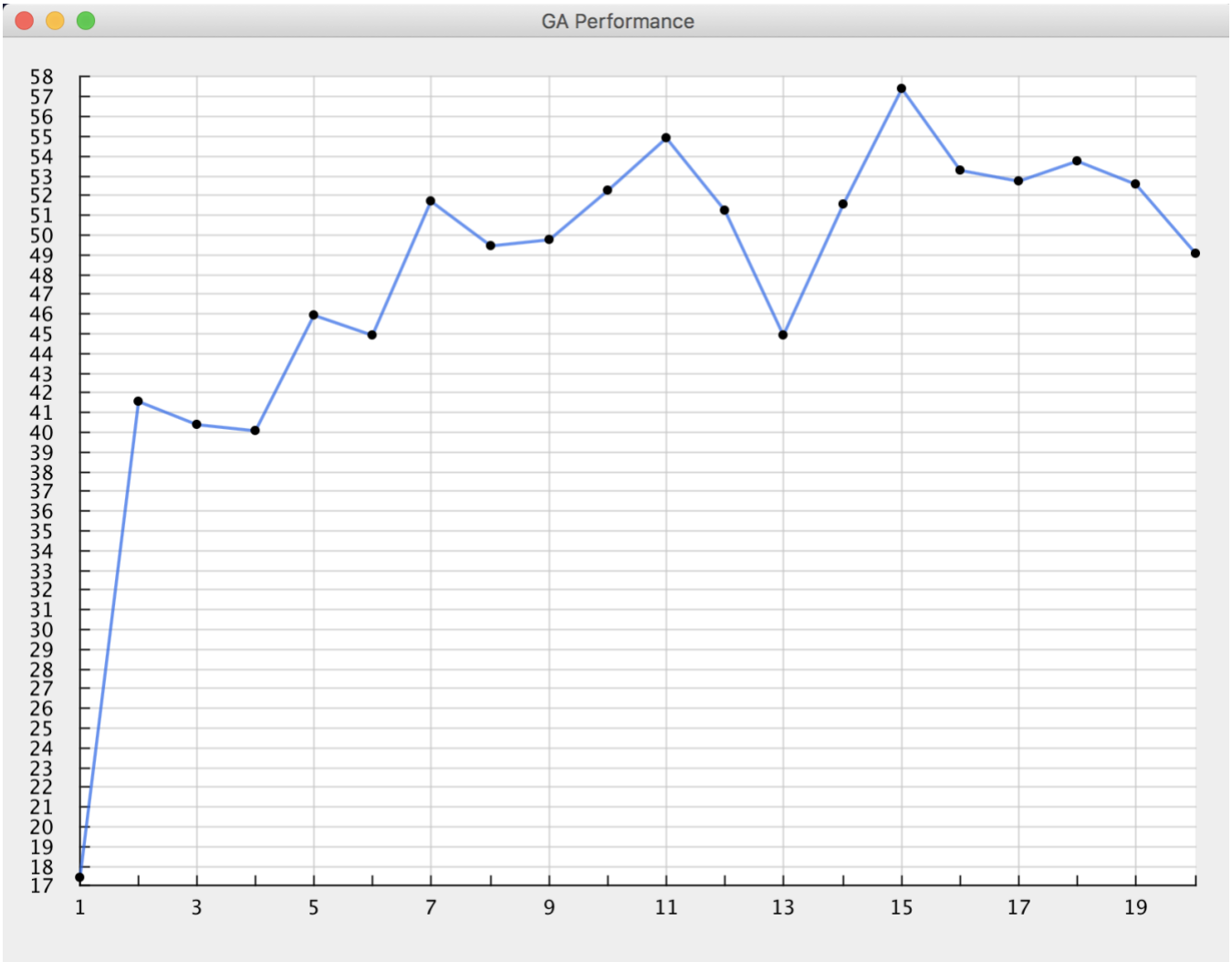
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
24	30	0.85	0.01	1,3	65	48.20833333333333

25.Combination 25



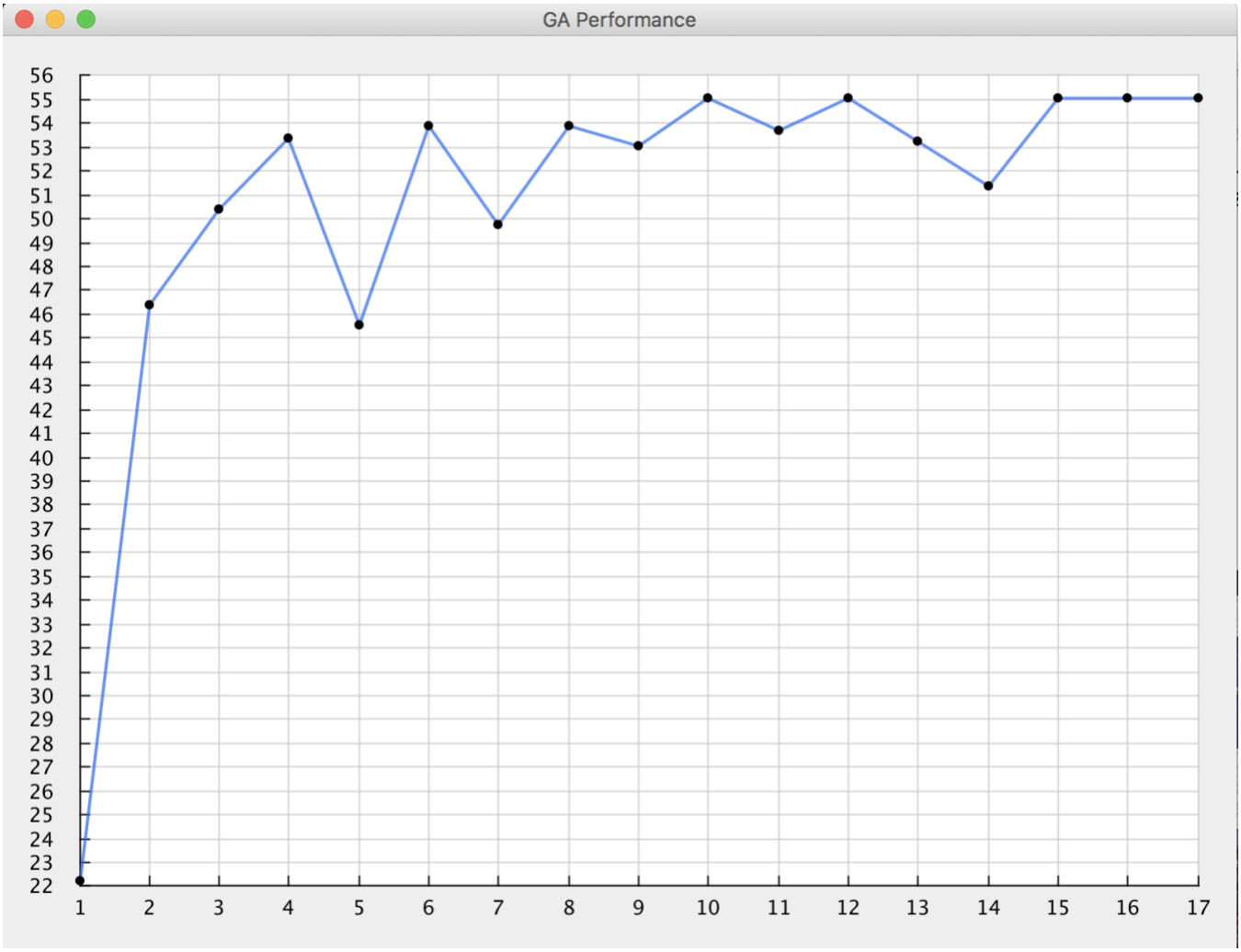
#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
25	30	0.95	0.1	1,3	65	54.26851851851851

26.Combination 26



#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
26	30	0.1	0.90	1,3	65	47.666666666666664

27.Combination 27



#	Population size	Crossover rate	Mutation rate	Items	Fitness	AVG Fitness
27	30	0.1	0.85	2,4	60	50.65686274509804

3 ANALYSIS

After testing the run using different population sizes, mutation rate and crossover rate values (Total of 3^3 combinations). We conclude that when we increase the crossover rate it will lead to the best average fitness function. On the other hand, if we decrease the crossover rate it will slowly reach the best average fitness function. Moreover, we conclude the lower value of mutation rate generates best average fitness function.