Project - Genetic Algorithms

Machine, Data and Learning

Tejasvi Chebrolu, Ruthvik Kodati

2019114005, 2019101035

Genetic Algorithm

A genetic algorithm is a metaheuristic inspired by the process of natural selection. These algorithms are commonly used to generate solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover, and selection.

In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) that can be mutated and altered.

The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimisation problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

Our Algorithm

Our algorithm is a modified version of the classic genetic algorithm with a stricter selection function to ensure we converge quickly to a lower MSE in fewer generations.

The flow of the genetic algorithm is shown below.

Step 1: Initialise New Generation
Step 2: Selection
Step 3: Crossover
Step 4: Mutation
Repeat

The following are the explanations for the sub-steps:

Fitness Function

```
# Test + Abs(val - test)
def fitness_func(errors):
    return  errors[1] + 0.5 * abs(errors[1] - errors[0])
```

The fitness function used is basically trying to minimize the sum of the validation error of the vector and half the positive difference of the training and validation error. This is based on the bias-variance tradeoff wherein we try to minimize both the variance and the bias. The best vector is one which has a lower fitness score.

Initial Population Creation

```
# Return a person
def init_mutation(person):
    temp = []
    for j in range(11):
        prob = np.random.randint(0, 100)
        if (prob < 15):
            dev = abs(person[j])/10 + random.uniform(-10, 10)/10
            if random.random() < 0.5:
                temp.append(person[j] + dev)
            else:
                temp.append(person[j] - dev)
            temp[j] = min(10, temp[j])
            temp[j] = max(-10, temp[j])
        else:
            temp.append(person[j])
    return temp
```

To initialize the population, we initially took a random number between 0 and 100. This basically represents a probability. If the probability was greater than 15, we did not make any mutation. Otherwise, we either added or subtracted the value `dev` (50% chance of either).

`dev` was defined as one tenth of the absolute of the initial value added with a random number between -1 and 1.

Selection

```
# Select five pairs of vectors
def selection(population):
    pop_len = len(population)
    errortuple = []
    for i in range(pop_len):
        errors = get_errors(key, population[i])
        theerrortuple = (fitness_func(errors), i)
        errortuple.append(theerrortuple)
    errortuple.sort(key = lambda x: x[0])

    prob = random.randint(0, 1)
    parents = []
    if prob == 0:
        parents.append((population[errortuple[0][1]], population[errortuple[1][1]]))
        parents.append((population[errortuple[0][1]], population[errortuple[2][1]]))
        parents.append((population[errortuple[0][1]], population[errortuple[3][1]]))
        parents.append((population[errortuple[1][1]], population[errortuple[2][1]]))
        parents.append((population[errortuple[1][1]], population[errortuple[3][1]]))
    else:
        parents.append((population[errortuple[1][1]], population[errortuple[0][1]]))
        parents.append((population[errortuple[1][1]], population[errortuple[2][1]]))
        parents.append((population[errortuple[1][1]], population[errortuple[3][1]]))
        parents.append((population[errortuple[0][1]], population[errortuple[2][1]]))
        parents.append((population[errortuple[0][1]], population[errortuple[3][1]]))
    return parents
```

Our selection function is the biggest change to the base genetic algorithm. We've opted for a very strict selection function that selects only the four best vectors in a population. The vectors were sorted according to the fitness function with the best vector occupying the zeroth index in `errortuple`.

Let the 4 best vectors in the population be *A*, *B*, *C*, and *D* respectively.

We either chose *Selection Process A* or *B* according to the value of `prob`. `prob` is either zero or one.

Selection Process

Pairs	Selection A	Selection B
1	**(A, B)**	**(B, A)**
2	**(A, C)**	**(B, C)**
3	**(A, D)**	**(B, D)**
4	**(B, C)**	**(A, C)**
5	**(B, D)**	**(A, D)**

Crossover

```
# Return a child
def crossover(male, female):
    temp = []
    for i in range(11):
        prob = random.randint(0, 1)
        if prob == 0:
            temp.append(male[i])
        else:
            temp.append(female[i])
    return temp
```

Our crossover function takes in two vectors as the input and returns a child. Each gene in the child has a 50% chance of coming from either parent. This is similar to how crossover happens in a traditional genetic algorithm. The crossover function is run twice for each pair of parents.

Mutations

```
def mutation(person):
    temp = []
    for j in range(11):
        prob = np.random.randint(0, 100)
        if (prob < 15):
            dev = abs(person[j])/10
            if random.random() < 0.5:
                temp.append(person[j] + dev)
            else:
                temp.append(person[j] - dev)
            temp[j] = min(10, temp[j])
            temp[j] = max(-10, temp[j])
        else:
            temp.append(person[j])
    return temp
```

To mutate a vector, we initially took a random number between 0 and 100. This basically represents a probability. If the probability was greater than 15, we did not make any mutation. Otherwise, we either added or subtracted the value `dev` (50% chance of either). `dev` was defined as one tenth of the absolute of the initial value.

This was done to ensure that the mutations occurring were not very large and that there was little chance of the generation diverging away from the required result.

Creation of New Generation

The vectors formed after the mutation were termed to be the new generation and were then used in the next iteration of the Genetic Algorithm. This process was then repeated multiple times to achieve better results. As the number of generations increased, a gradual decrease in the fitness scores was observed.

Three Iterations of Our Algorithm

Iteration 1:

The first iteration creates the generation 66 which is denoted by the vectors K-T in the first diagram.

Iteration 2:

The second iteration creates the generation 67 which is denoted by the vectors K-T in the second diagram.

Iteration 3:

The final iteration creates the generation 68 which is denoted by the vectors K-T in the last diagram.

1

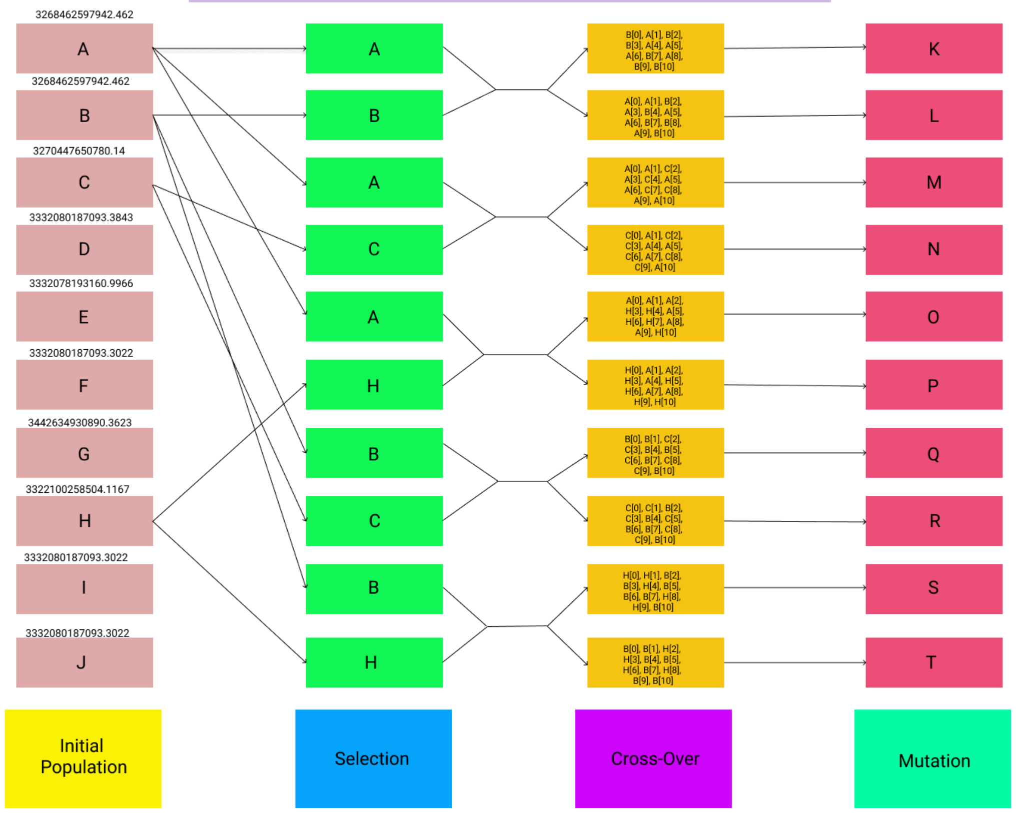
A-J vectors are found in generations 65.

K-T vectors are found in generations 66.

```
A = [
  -5.613299999999998e-21,
  0.6301658485657119,
  -5.378780858e-13,
  1.7821182830000003e-11,
  -2.7633629429999994e-10,
  -3.839626233000001e-16,
  1.070238466e-15,
  8.7556396085983e-06,
  -3.4752413629999996e-07,
  -2.79792834e-08,
  6.082140340000001e-10
 ]

B = [
  -5.613299999999998e-21,
  0.6301658485657119,
  -5.378780858e-13,
  1.7821182830000003e-11,
  -2.7633629429999994e-10,
  -3.839626233000001e-16,
  1.070238466e-15,
  8.7556396085983e-06,
  -3.4752413629999996e-07,
  -2.79792834e-08,
  6.082140340000001e-10
 ]

C =  [
  -6.236999999999998e-21,
  0.6301658485657119,
  -5.378780858e-13,
  1.7821182830000003e-11,
  -2.7633629429999994e-10,
  -4.266251370000001e-16,
  8.756496540000001e-16,
  8.7556396085983e-06,
  -3.4752413629999996e-07,
  -2.79792834e-08,
  6.082140340000001e-10
 ]

D =  [
  -6.236999999999998e-21,
  0.6301658485657119,
  -5.378780858e-13,
  1.7821182830000003e-11,
  -2.7633629429999994e-10,
  -4.266251370000001e-16,
  9.7294406e-16,
  7.959672371453e-06,
  -3.4752413629999996e-07,
  -2.79792834e-08,
  6.082140340000001e-10
 ]

E = [
  -6.236999999999998e-21,
  0.6931824334222831,
  -5.378780858e-13,
  1.9603301113000004e-11,
  -2.4870266486999994e-10,
  -4.266251370000001e-16,
  1.070238466e-15,
  7.959672371453e-06,
  -3.4752413629999996e-07,
  -2.79792834e-08,
  6.082140340000001e-10
 ]

F = [
  -6.236999999999998e-21,
  0.6301658485657119,
  -5.378780858e-13,
  1.9603301113000004e-11,
  -2.7633629429999994e-10,
  -4.266251370000001e-16,
  9.7294406e-16,
  7.959672371453e-06,
  -3.4752413629999996e-07,
  -2.79792834e-08,
  6.082140340000001e-10
 ]

G = [
  -5.051969999999998e-21,
  0.6301658485657119,
  -5.378780858e-13,
  1.7821182830000003e-11,
  -2.7633629429999994e-10,
  -3.839626233000001e-16,
  1.070238466e-15,
  7.1637051343077e-06,
  -3.4752413629999996e-07,
  -2.79792834e-08,
  6.082140340000001e-10
 ]

H =  [
  -5.613299999999998e-21,
  0.6301658485657119,
  -5.378780858e-13,
  1.7821182830000003e-11,
  -2.7633629429999994e-10,
  -3.839626233000001e-16,
  1.070238466e-15,
  7.959672371453e-06,
  -3.4752413629999996e-07,
  -2.79792834e-08,
  6.082140340000001e-10
 ]

I =  [
  -6.236999999999998e-21,
  0.6301658485657119,
  -5.378780858e-13,
  1.9603301113000004e-11,
  -2.7633629429999994e-10,
  -4.266251370000001e-16,
  1.070238466e-15,
  7.959672371453e-06,
  -3.4752413629999996e-07,
  -2.79792834e-08,
  6.082140340000001e-10
 ]

J =  [
  -6.236999999999998e-21,
  0.6301658485657119,
  -5.378780858e-13,
  1.9603301113000004e-11,
  -2.7633629429999994e-10,
  -4.266251370000001e-16,
  1.070238466e-15,
  7.959672371453e-06,
  -3.4752413629999996e-07,
  -2.79792834e-08,
  6.082140340000001e-10
 ]
```

```
K = [
  -6.1746299999999984e-21,
  0.6301658485657119,
  -5.378780858e-13,
  1.7821182830000003e-11,
  -2.4870266486999994e-10,
  -3.839626233000001e-16,
  1.070238466e-15,
  8.7556396085983e-06,
  -3.1277172266999995e-07,
  -3.077721174e-08,
  6.082140340000001e-10
 ]

L = [
  -5.613299999999998e-21,
  0.6301658485657119,
  -5.378780858e-13,
  1.7821182830000003e-11,
  -2.7633629429999994e-10,
  -3.839626233000001e-16,
  1.070238466e-15,
  8.7556396085983e-06,
  -3.4752413629999996e-07,
  -2.79792834e-08,
  6.082140340000001e-10
 ]

M = [
  -6.236999999999998e-21,
  0.6301658485657119,
  -5.9166589438e-13,
  1.7821182830000003e-11,
  -2.7633629429999994e-10,
  -4.266251370000001e-16,
  1.1772623126e-15,
  7.880075647738471e-06,
  -3.4752413629999996e-07,
  -2.79792834e-08,
  6.082140340000001e-10
 ]

N = [
  -6.236999999999998e-21,
  0.6301658485657119,
  -5.378780858e-13,
  1.7821182830000003e-11,
  -2.7633629429999994e-10,
  -4.266251370000001e-16,
  1.070238466e-15,
  8.7556396085983e-06,
  -3.4752413629999996e-07,
  -2.79792834e-08,
  6.082140340000001e-10
 ]

O = [
  -5.051969999999998e-21,
  0.6301658485657119,
  -4.8409027722e-13,
  1.7821182830000003e-11,
  -2.7633629429999994e-10,
  -3.839626233000001e-16,
  1.070238466e-15,
  7.959672371453e-06,
  -3.4752413629999996e-07,
  -2.79792834e-08,
  6.690354374000001e-10
 ]

P =  [
  -5.613299999999998e-21,
  0.6301658485657119,
  -5.378780858e-13,
  1.7821182830000003e-11,
  -2.7633629429999994e-10,
  -3.839626233000001e-16,
  1.070238466e-15,
  7.959672371453e-06,
  -3.4752413629999996e-07,
  -2.79792834e-08,
  6.082140340000001e-10
 ]

Q = [
  -6.236999999999998e-21,
  0.5671492637091406,
  -5.378780858e-13,
  1.7821182830000003e-11,
  -2.7633629429999994e-10,
  -3.839626233000001e-16,
  1.070238466e-15,
  8.7556396085983e-06,
  -3.1277172266999995e-07,
  -2.79792834e-08,
  6.082140340000001e-10
 ]

R =  [
  -6.236999999999998e-21,
  0.6301658485657119,
  -5.378780858e-13,
  1.7821182830000003e-11,
  -2.7633629429999994e-10,
  -4.266251370000001e-16,
  1.070238466e-15,
  8.7556396085983e-06,
  -3.4752413629999996e-07,
  -2.79792834e-08,
  6.082140340000001e-10
 ]

S =  [
  -5.613299999999998e-21,
  0.6301658485657119,
  -5.378780858e-13,
  1.7821182830000003e-11,
  -2.7633629429999994e-10,
  -3.839626233000001e-16,
  9.632146194e-16,
  8.7556396085983e-06,
  -3.4752413629999996e-07,
  -2.518135506e-08,
  5.473926306e-10
 ]

T = [
  -5.613299999999998e-21,
  0.6301658485657119,
  -5.378780858e-13,
  1.7821182830000003e-11,
  -2.7633629429999994e-10,
  -3.839626233000001e-16,
  1.070238466e-15,
  8.7556396085983e-06,
  -3.4752413629999996e-07,
  -2.79792834e-08,
  6.082140340000001e-10
 ]
```

(fitnesses are mentioned above the initial population vectors)



3268462597942.462
A
3268462597942.462
B
3270447650780.14
C
3332080187093.3843
D
3332078193160.9966
E
3332080187093.3022
F
3442634930890.3623
G
3322100258504.1167
H
3332080187093.3022
I
3332080187093.3022
J
A
B
A
C
A
H
B
C
B
H
B[0], A[1], B[2], B[3], A[4], A[5], A[6], B[7], A[8], B[9], B[10]
A[0], A[1], B[2], A[3], B[4], A[5], A[6], B[7], B[8], A[9], B[10]
A[0], A[1], C[2], A[3], C[4], A[5], A[6], C[7], C[8], A[9], A[10]
C[0], A[1], C[2], C[3], A[4], A[5], C[6], A[7], C[8], C[9], A[10]
A[0], A[1], A[2], H[3], H[4], A[5], H[6], H[7], A[8], A[9], H[10]
H[0], A[1], A[2], H[3], A[4], H[5], H[6], A[7], A[8], H[9], H[10]
B[0], B[1], C[2], C[3], B[4], B[5], C[6], B[7], C[8], C[9], B[10]
C[0], C[1], B[2], C[3], B[4], C[5], B[6], B[7], C[8], C[9], B[10]
H[0], H[1], B[2], B[3], H[4], B[5], B[6], B[7], H[8], H[9], B[10]
B[0], B[1], H[2], H[3], B[4], B[5], H[6], B[7], H[8], B[9], B[10]
K
L
M
N
O
P
Q
R
S
T
Initial Population
Selection
Cross-Over
Mutation

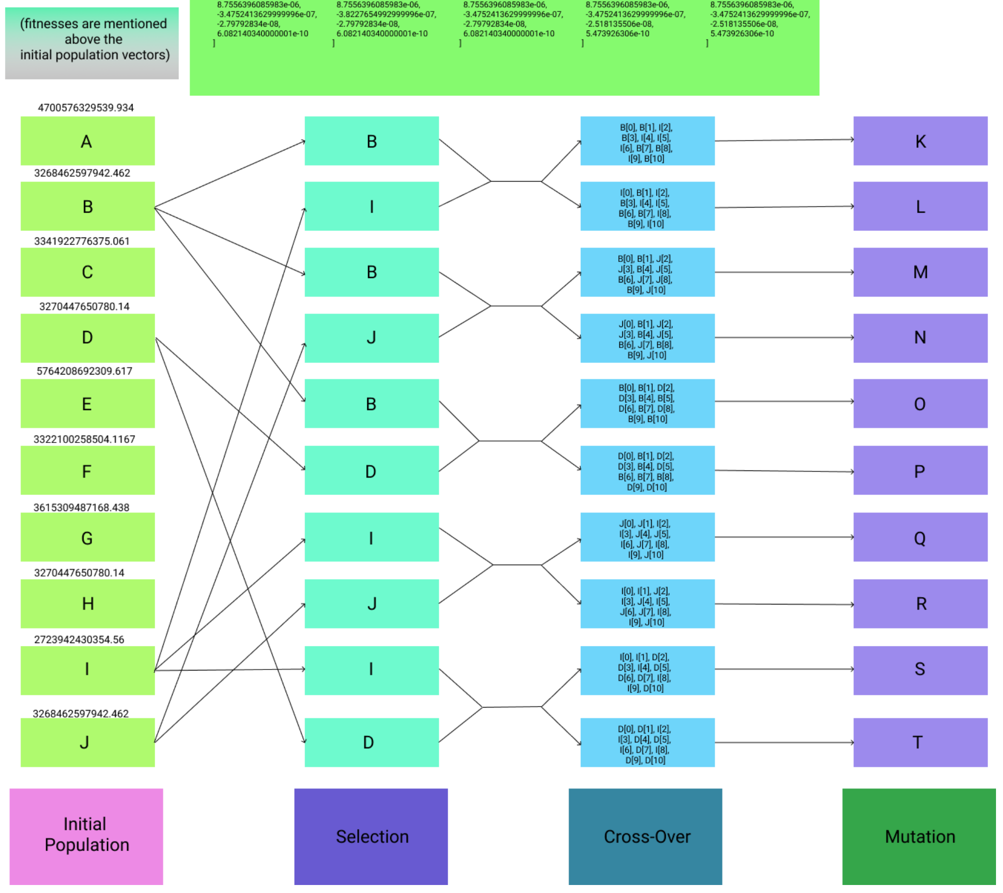
2

A-J vectors are found in generations 66.

K-T vectors are found in generations 67.

```
A = [
   -6.1746299999999984e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.4870266486999994e-10,
   -3.839626233000001e-16,
   1.070238466e-15,
   8.7556396085983e-06,
   -3.1277172266999995e-07,
   -3.077721174e-08,
   6.082140340000001e-10
 ]

B = [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   1.070238466e-15,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

C =   [
   -6.2369999999999998e-21,
   0.6301658485657119,
   -5.9166589438e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -4.266251370000001e-16,
   1.1772623126e-15,
   7.880075647738471e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

D =   [
   -6.2369999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -4.266251370000001e-16,
   1.070238466e-15,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

E = [
   -5.0519699999999998e-21,
   0.6301658485657119,
   -4.8409027722e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   1.070238466e-15,
   7.959672371453e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.690354374000001e-10
 ]

F = [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   1.070238466e-15,
   7.959672371453e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

G =  [
   -6.2369999999999998e-21,
   0.5671492637091406,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   1.070238466e-15,
   8.7556396085983e-06,
   -3.1277172266999995e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

H =  [
   -6.2369999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -4.266251370000001e-16,
   1.070238466e-15,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

I =  [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   9.632146194e-16,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.518135506e-08,
   5.473926306e-10
 ]

J =   [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   1.070238466e-15,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]
```

```
K = [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.9166589438e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   1.070238466e-15,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.518135506e-08,
   4.926533675400001e-10
 ]

L = [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   1.070238466e-15,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   5.473926306e-10
 ]

M = [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   1.070238466e-15,
   9.63120356945813e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

N = [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   1.070238466e-15,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

O = [
   -6.2369999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -4.266251370000001e-16,
   1.1772623126e-15,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

P =  [
   -6.2369999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -4.266251370000001e-16,
   1.070238466e-15,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

Q =  [
   -5.6132999999999998e-21,
   0.6931824334222831,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   9.632146194e-16,
   8.7556396085983e-06,
   -3.8227654992999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

R =  [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   9.632146194e-16,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

S =   [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.6039064547000002e-11,
   -2.7633629429999994e-10,
   -4.266251370000001e-16,
   9.632146194e-16,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.518135506e-08,
   5.473926306e-10
 ]

T = [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -4.266251370000001e-16,
   9.632146194e-16,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.518135506e-08,
   5.473926306e-10
 ]
```



(fitnesses are mentioned above the initial population vectors)
4700576329539.934
A
3268462597942.462
B
3341922776375.061
C
3270447650780.14
D
5764208692309.617
E
3322100258504.1167
F
3615309487168.438
G
3270447650780.14
H
2723942430354.56
I
3268462597942.462
J
B
I
B
J
B
D
I
J
I
D
B[0], B[1], I[2], B[3], I[4], I[5], I[6], B[7], B[8], I[9], B[10]
I[0], B[1], I[2], B[3], I[4], I[5], B[6], B[7], I[8], B[9], I[10]
B[0], B[1], J[2], J[3], B[4], J[5], B[6], J[7], J[8], B[9], J[10]
J[0], B[1], J[2], J[3], B[4], J[5], B[6], J[7], B[8], B[9], J[10]
B[0], B[1], D[2], D[3], B[4], B[5], D[6], B[7], D[8], B[9], B[10]
D[0], B[1], D[2], D[3], B[4], D[5], B[6], B[7], B[8], D[9], D[10]
J[0], J[1], I[2], I[3], J[4], J[5], I[6], J[7], I[8], I[9], J[10]
I[0], I[1], J[2], I[3], J[4], I[5], J[6], J[7], I[8], I[9], J[10]
I[0], I[1], D[2], D[3], I[4], D[5], D[6], D[7], I[8], I[9], D[10]
D[0], D[1], I[2], I[3], D[4], D[5], I[6], D[7], I[8], D[9], D[10]
K
L
M
N
O
P
Q
R
S
T
Initial Population
Selection
Cross-Over
Mutation

3

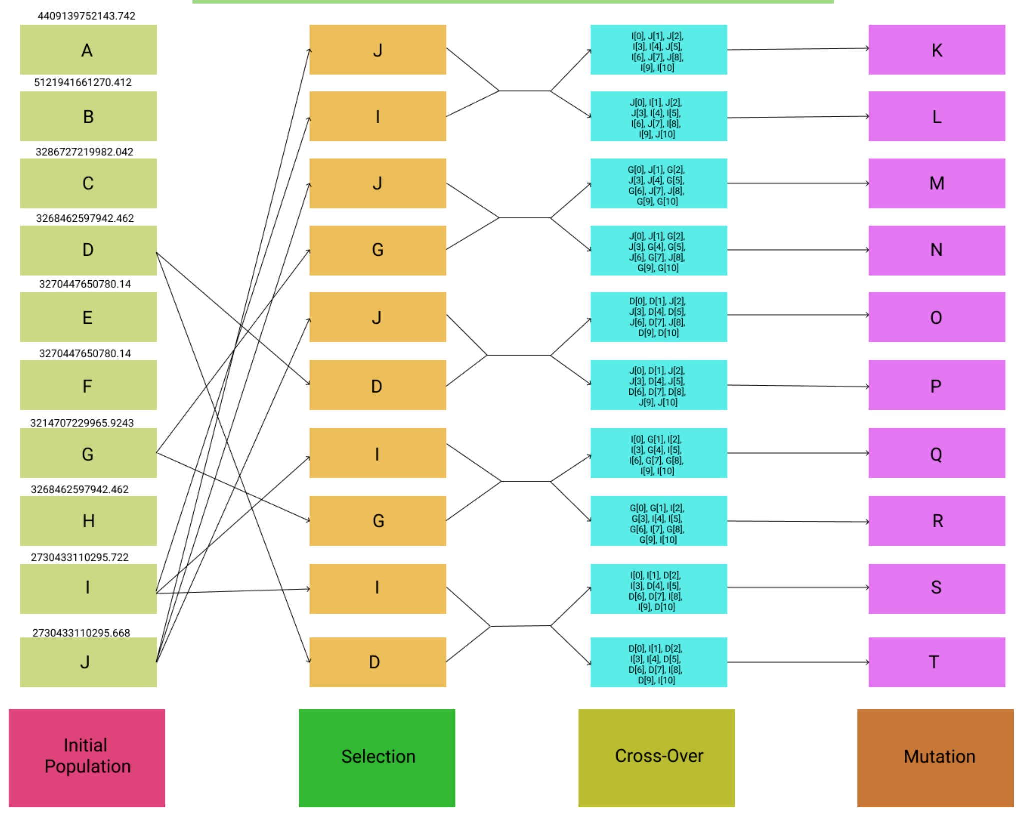
A-J vectors are found in generations 67.

K-T vectors are found in generations 68.

```
A = [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.9166589438e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   1.070238466e-15,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.518135506e-08,
   4.926533675400001e-10
 ],

B = [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   1.070238466e-15,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   5.473926306e-10
 ]

C =  [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   1.070238466e-15,
   9.63120356945813e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

D =  [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   1.070238466e-15,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

E =[
   -6.2369999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -4.266251370000001e-16,
   1.1772623126e-15,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

F =  [
   -6.2369999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -4.266251370000001e-16,
   1.070238466e-15,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

G = [
   -5.6132999999999998e-21,
   0.6931824334222831,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   9.632146194e-16,
   8.7556396085983e-06,
   -3.8227654992999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

H = [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   9.632146194e-16,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

I =  [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.6039064547000002e-11,
   -2.7633629429999994e-10,
   -4.266251370000001e-16,
   9.632146194e-16,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.518135506e-08,
   5.473926306e-10
 ]

J =   [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -4.266251370000001e-16,
   9.632146194e-16,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.518135506e-08,
   5.473926306e-10
 ]
```

```
K =  [
   -5.0519699999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.6039064547000002e-11,
   -2.4870266486999994e-10,
   -4.692876507000002e-16,
   9.632146194e-16,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.518135506e-08,
   5.473926306e-10
 ]

L =[
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.6039064547000002e-11,
   -2.7633629429999994e-10,
   -4.266251370000001e-16,
   9.632146194e-16,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.518135506e-08,
   5.473926306e-10
 ]

M = [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -4.266251370000001e-16,
   9.632146194e-16,
   7.880075647738471e-06,
   -3.4752413629999996e-07,
   -3.077721174e-08,
   6.082140340000001e-10
 ]

N = [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -4.266251370000001e-16,
   9.632146194e-16,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

O =[
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.6039064547000002e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   9.632146194e-16,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

P = [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.6039064547000002e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   9.632146194e-16,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

Q =  [
   -5.6132999999999998e-21,
   0.6931824334222831,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   9.632146194e-16,
   8.7556396085983e-06,
   -3.8227654992999996e-07,
   -2.518135506e-08,
   6.082140340000001e-10
 ]

R = [
   -5.6132999999999998e-21,
   0.6931824334222831,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -4.266251370000001e-16,
   9.632146194e-16,
   8.7556396085983e-06,
   -3.8227654992999996e-07,
   -2.518135506e-08,
   6.082140340000001e-10
 ]

S =   [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -3.0396992372999994e-10,
   -3.839626233000001e-16,
   1.070238466e-15,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]

T = [
   -5.6132999999999998e-21,
   0.6301658485657119,
   -5.378780858e-13,
   1.7821182830000003e-11,
   -2.7633629429999994e-10,
   -3.839626233000001e-16,
   1.070238466e-15,
   8.7556396085983e-06,
   -3.4752413629999996e-07,
   -2.79792834e-08,
   6.082140340000001e-10
 ]
```

(fitnesses are mentioned above the initial population vectors)



4409139752143.742
A
5121941661270.412
B
3286727219982.042
C
3268462597942.462
D
3270447650780.14
E
3270447650780.14
F
3214707229965.9243
G
3268462597942.462
H
2730433110295.722
I
2730433110295.668
J
J
I
J
G
J
D
I
G
I
D
I[0], J[1], J[2], I[3], I[4], J[5], I[6], J[7], J[8], I[9], I[10]
J[0], I[1], J[2], J[3], I[4], I[5], I[6], J[7], I[8], I[9], J[10]
G[0], J[1], G[2], J[3], J[4], G[5], G[6], J[7], J[8], G[9], G[10]
J[0], J[1], G[2], J[3], G[4], G[5], J[6], G[7], J[8], G[9], G[10]
D[0], D[1], J[2], J[3], D[4], D[5], J[6], D[7], J[8], D[9], D[10]
J[0], D[1], J[2], J[3], D[4], J[5], D[6], D[7], D[8], J[9], J[10]
I[0], G[1], I[2], I[3], G[4], I[5], I[6], G[7], G[8], I[9], I[10]
G[0], G[1], I[2], G[3], I[4], I[5], G[6], I[7], G[8], G[9], I[10]
I[0], I[1], D[2], I[3], D[4], I[5], D[6], D[7], I[8], I[9], D[10]
D[0], I[1], D[2], I[3], I[4], D[5], D[6], D[7], I[8], D[9], I[10]
K
L
M
N
O
P
Q
R
S
T
Initial Population
Selection
Cross-Over
Mutation

Hyperparameters

The final hyperparameters used in our algorithm are as follows -

- **Pool Size**
    - The population size in a generation was 10. Therefore, the mating pool size was 5. This was selected because we wanted a population size that was big enough to see some amount of variation, but small enough to manually inspect the population when needed.
- **Splitting Point**
    - Our algorithm has a different take on the regular crossover function. Instead of splitting the genes into two and recombining, we take the genes directly from the parents. However, there is a 50% chance that a particular gene comes from either parent.

Statistical Information

Over the course of the project, we had different variations to the base algorithm which resulted in different number of iterations required for the algorithm to converge. The initial algorithm, where the probability of mutation was very high(25%), took around 35-40 generations to reach a plateau. Looking at this, we decided to reduce the probability of mutation to 15% and it took around 60 generations to converge to the final result that we generated.

Similarly, while making changes to mutation, we also changed the amount of mutation and realised that higher mutation was leading to more variance and resulted in the generations drifting apart to nowhere.

Heuristics

The different heuristics tried in our algorithm are shown below:

- **Mutation**
    - There were two kinds of changes made to the mutation function. The first kind was regarding the percentage chance of a mutation happening. This was initially

set to a high percentage but then was reduced to 15% to ensure little divergence and more convergence. The second change was with respect to the amount of mutation done. Initially, we were making very high mutations as we were adding numbers in the same range as the original gene but we realised for better results, it should be a lower number that needs to be added.

- ▼ **Crossover**
  - ▼ The crossover function is extremely novel and different from the traditional function used in regular genetic algorithms. This was modelled based on real-life wherein a child receives the same genes as that of a parent.
- ▼ **Selection**
  - ▼ The selection of the mating pool was done in such a way that only the best vectors in a particular generation would mate. To ensure some amount of randomness, we introduced an element of a coin toss as to what kind of Selection would be followed(*A* or *B* as seen in the selection section above). This was to ensure that we could converge quickly and at the same time, to make sure that only the best vectors are mating which would lead to better results.

Errors

The initial overfit vector given to us had a training error of around *1e10* and a validation error of around *3e11*. The objective of the project was to reduce the validation error to around the same as the training error while ensuring that the training error remained approximately the same. This was done because we wanted to achieve the bias-variance tradeoff. The best vectors that were selected are the ones that have their training error approximately the same, while reducing the validation errors.

These low training and validation errors mean that we have managed to somewhat reduce the validation error while making sure that the training error is also low. This is keeping in mind the bias-variance tradeoff. These vectors will now work well on unseen data because the difference between the training error and validation error is low. Moreover, we have reduced the overfitting and made sure that it follows no particular function.

Miscellaneous

There was very little brute force in the algorithms that we ran. The only times we made any changes were making small mutations to any vector that we thought had an exceptionally low bias and variance. This was done using the `init_mutation` function that makes mutations off of a particular vector. This was done to get a better rank on the leaderboard.
