Machine Data and Learning Project: Genetic Algorithm
Team 77: Matrix Monarchs

Ahana Datta - 2019111007
Tanvi Narsapur-2019111005

Basic Overview of the Algorithm:

The genetic algorithm follows the steps -

1. Initial population:
   The initial population is generated by mutating the given initial weight vector. The mutation probability is kept 0.9. The initial weight vector is also included in the initial population after mutating with a probability of 0.4.

2. Obtaining error for initial population:
   Using the "get_errors" function, the error values for each vector are generated by making requests to the server. This is done in the function "get_pop_errors". Based on the error values, the value of the fitness function is calculated and the elements of the initial population are sorted such that the element with the least error appears first. This sorting is done in the "sort_arrays" function.

3. Choosing parents:
   For selecting the parent indices, the probability of selection of an element as a parent is set using the Russian Roulette method. The thresholds are set in the function "russian_roulette". Based on these thresholds or probabilities, parents are chosen by generating a random number and comparing it with the probability. The selection of parents is done in the "get_parent_indices" function.

4. Crossover:
   The crossover of the chosen parents is performed using the Simulated Binary crossover method. (Population size)/2 parent pairs undergo crossover to create a child population of size equal to the population size. The child population is mutated with mutation probability 0.5. The mutation is performed by the "mutate" function.

5. Obtaining error for child population:
   The error for the child population is obtained using the "get_errors" function, based on which the fitness function value for each child element is calculated and the child population is sorted based on the value of the fitness function.

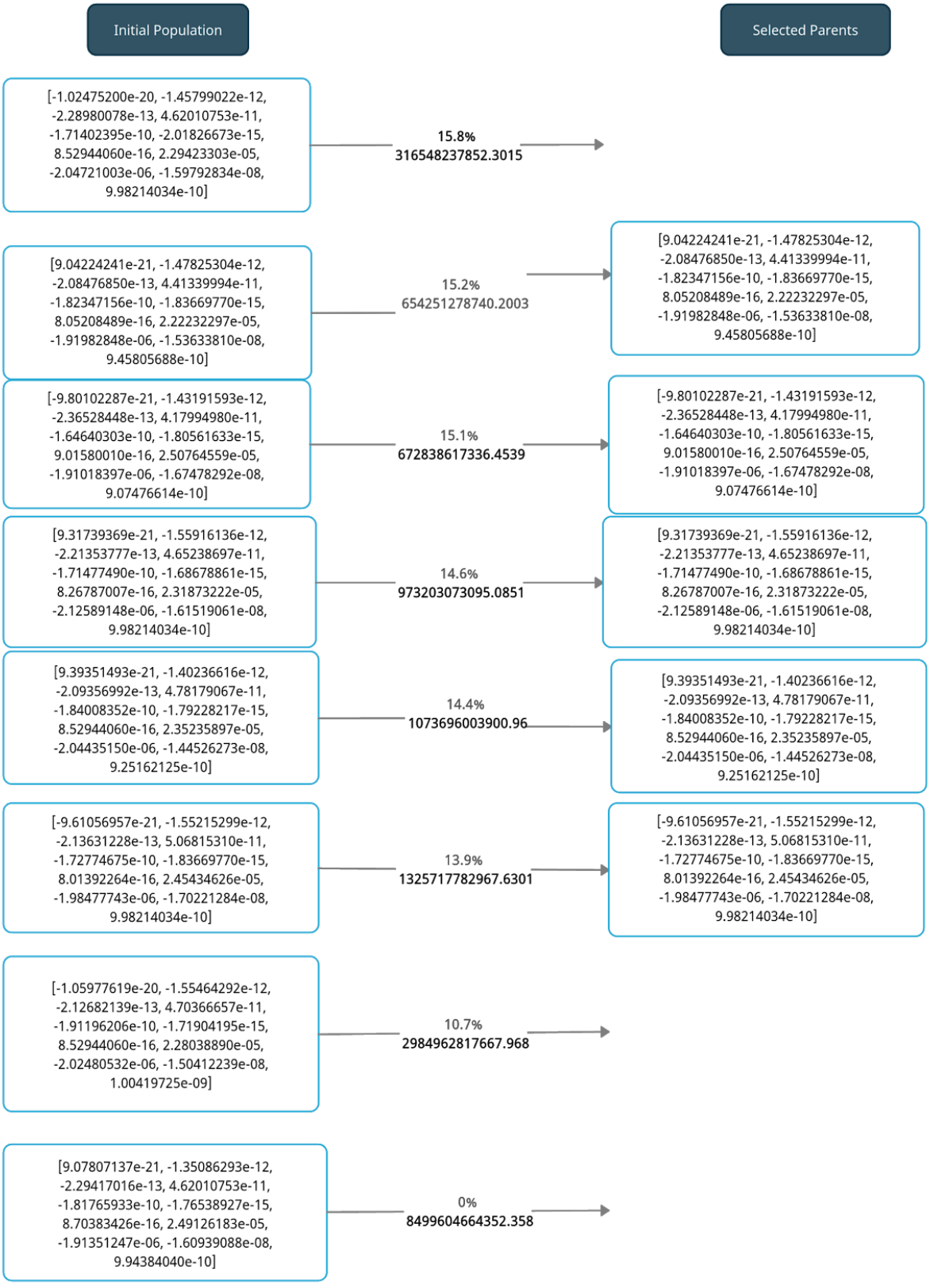
6. Forming the next generation population:
   For the next generation, the top n number of elements from the parent population and the top n number of elements from the child population is included in the next generation. For choosing the remaining elements in the next generation, the remaining elements in the parent and child population are clubbed together, sorted based on the fitness function value and finally, the top (population size - 2*n) elements are chosen from the clubbed population. And based on the fitness function we sort the newly created population.

7. Finding the minimum possible error:
   If the fitness function has a lower value as compared to the best vector found till now, then the best vector and the minimum value of the error is updated.

8. Executing the genetic algorithm:
   Steps 3 to 7 are repeated for gen number of iterations, that is in one run gen number of generations are generated (this was implemented because a fixed number of requests to the server were allowed per day.)

Diagrams:

Diagram 1 -



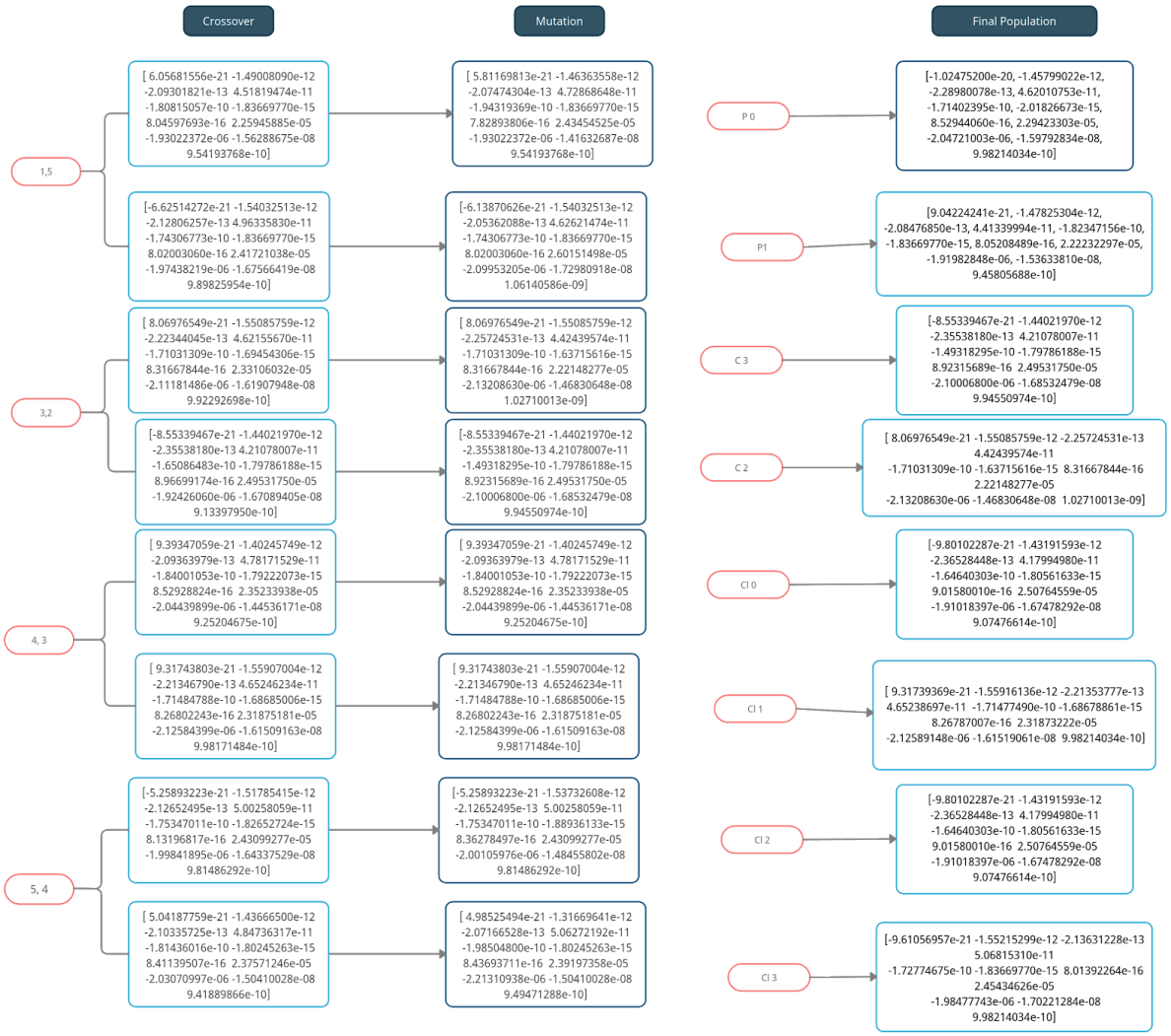
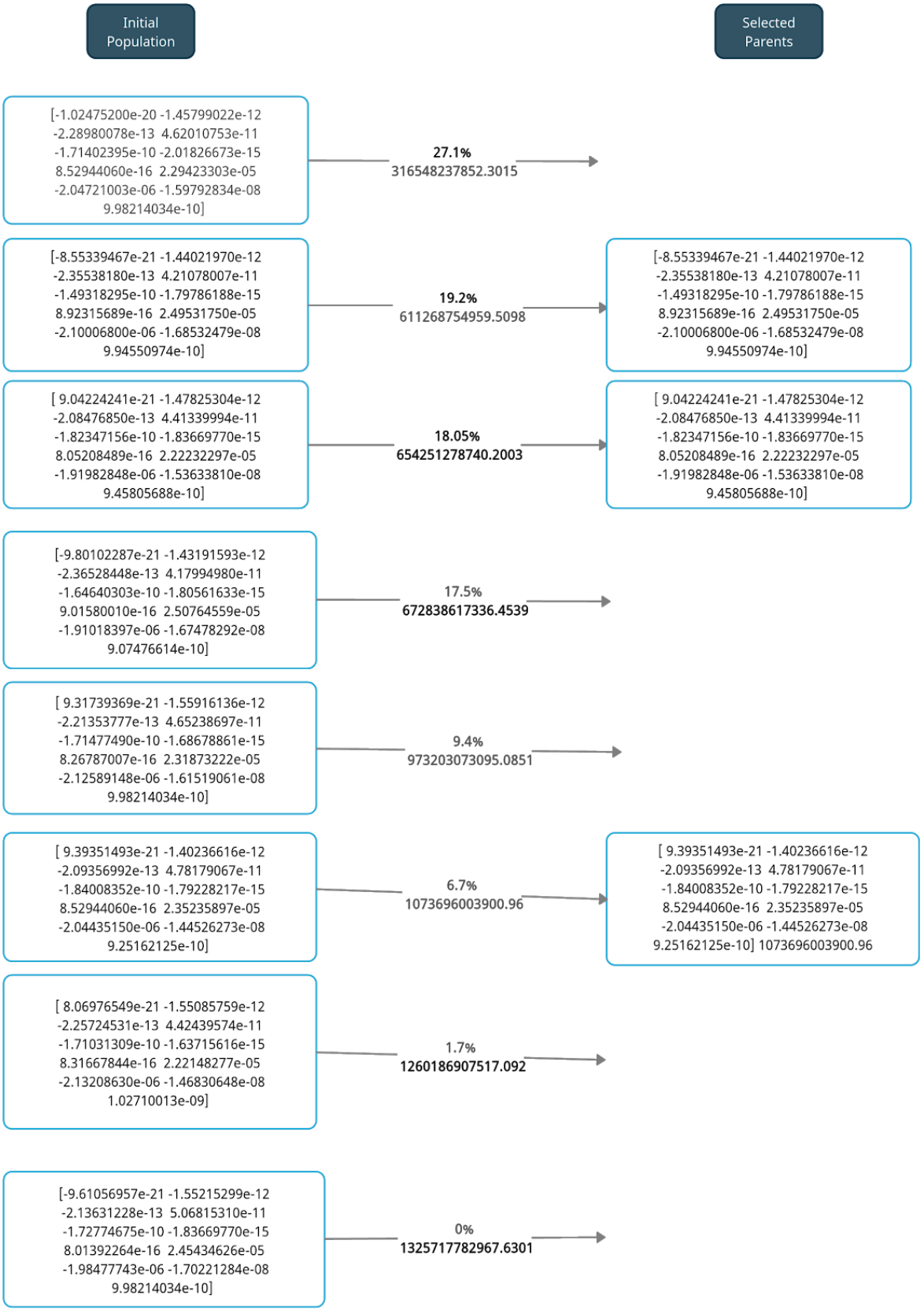

Crossover
Mutation
Final Population
1,5
[6.05681556e-21 -1.49008090e-12
-2.09301821e-13 4.51819474e-11
-1.80815057e-10 -1.83669770e-15
8.04597693e-16 2.25945885e-05
-1.93022372e-06 -1.56288675e-08
9.54193768e-10]
[5.81169813e-21 -1.46363558e-12
-2.07474304e-13 4.72868648e-11
-1.94319369e-10 -1.83669770e-15
7.82893806e-16 2.43454525e-05
-1.93022372e-06 -1.41632687e-08
9.54193768e-10]
[-6.62514272e-21 -1.54032513e-12
-2.12806257e-13 4.96335830e-11
-1.74306773e-10 -1.83669770e-15
8.02003060e-16 2.41721038e-05
-1.97438219e-06 -1.67566419e-08
9.89825954e-10]
[-6.13870626e-21 -1.54032513e-12
-2.05362088e-13 4.62621474e-11
-1.74306773e-10 -1.83669770e-15
8.02003060e-16 2.60151498e-05
-2.09953205e-06 -1.72980918e-08
1.06140586e-09]
3,2
[8.06976549e-21 -1.55085759e-12
-2.22344045e-13 4.62155670e-11
-1.71031309e-10 -1.69454306e-15
8.31667844e-16 2.33106032e-05
-2.11181486e-06 -1.61907948e-08
9.92292698e-10]
[8.06976549e-21 -1.55085759e-12
-2.25724531e-13 4.42439574e-11
-1.71031309e-10 -1.63715616e-15
8.31667844e-16 2.22148277e-05
-2.13208630e-06 -1.46830648e-08
1.02710013e-09]
[-8.55339467e-21 -1.44021970e-12
-2.35538180e-13 4.21078007e-11
-1.65086483e-10 -1.79786188e-15
8.96699174e-16 2.49531750e-05
-1.92426060e-06 -1.67089405e-08
9.13397950e-10]
[-8.55339467e-21 -1.44021970e-12
-2.35538180e-13 4.21078007e-11
-1.49318295e-10 -1.79786188e-15
8.92315689e-16 2.49531750e-05
-2.10006800e-06 -1.68532479e-08
9.94550974e-10]
4, 3
[9.39347059e-21 -1.40245749e-12
-2.09363979e-13 4.78171529e-11
-1.84001053e-10 -1.79222073e-15
8.52928824e-16 2.35233938e-05
-2.04439899e-06 -1.44536171e-08
9.25204675e-10]
[9.39347059e-21 -1.40245749e-12
-2.09363979e-13 4.78171529e-11
-1.84001053e-10 -1.79222073e-15
8.52928824e-16 2.35233938e-05
-2.04439899e-06 -1.44536171e-08
9.25204675e-10]
[9.31743803e-21 -1.55907004e-12
-2.21346790e-13 4.65246234e-11
-1.71484788e-10 -1.68685006e-15
8.26802243e-16 2.31875181e-05
-2.12584399e-06 -1.61509163e-08
9.98171484e-10]
[9.31743803e-21 -1.55907004e-12
-2.21346790e-13 4.65246234e-11
-1.71484788e-10 -1.68685006e-15
8.26802243e-16 2.31875181e-05
-2.12584399e-06 -1.61509163e-08
9.98171484e-10]
5, 4
[-5.25893223e-21 -1.51785415e-12
-2.12652495e-13 5.00258059e-11
-1.75347011e-10 -1.82652724e-15
8.13196817e-16 2.43099277e-05
-1.99841895e-06 -1.64337529e-08
9.81486292e-10]
[-5.25893223e-21 -1.53732608e-12
-2.12652495e-13 5.00258059e-11
-1.75347011e-10 -1.88936133e-15
8.36278497e-16 2.43099277e-05
-2.00105976e-06 -1.48455802e-08
9.81486292e-10]
[5.04187759e-21 -1.43666500e-12
-2.10335725e-13 4.84736317e-11
-1.81436016e-10 -1.80245263e-15
8.41139507e-16 2.37571246e-05
-2.03070997e-06 -1.50410028e-08
9.41889866e-10]
[4.98525494e-21 -1.31669641e-12
-2.07166528e-13 5.06272192e-11
-1.98504800e-10 -1.80245263e-15
8.43693711e-16 2.39197358e-05
-2.21310938e-06 -1.50410028e-08
9.49471288e-10]
P 0
[-1.02475200e-20, -1.45799022e-12,
-2.28980078e-13, 4.62010753e-11,
-1.71402395e-10, -2.01826673e-15,
8.52944060e-16, 2.29423303e-05,
-2.04721003e-06, -1.59792834e-08,
9.98214034e-10]
P1
[9.04224241e-21, -1.47825304e-12,
-2.08476850e-13, 4.41339994e-11, -1.82347156e-10,
-1.83669770e-15, 8.05208489e-16, 2.22232297e-05,
-1.91982848e-06, -1.53633810e-08,
9.45805688e-10]
C 3
[-8.55339467e-21 -1.44021970e-12
-2.35538180e-13 4.21078007e-11
-1.49318295e-10 -1.79786188e-15
8.92315689e-16 2.49531750e-05
-2.10006800e-06 -1.68532479e-08
9.94550974e-10]
C 2
[8.06976549e-21 -1.55085759e-12 -2.25724531e-13
4.42439574e-11
-1.71031309e-10 -1.63715616e-15 8.31667844e-16
2.22148277e-05
-2.13208630e-06 -1.46830648e-08 1.02710013e-09]
Cl 0
[-9.80102287e-21 -1.43191593e-12
-2.36528448e-13 4.17994980e-11
-1.64640303e-10 -1.80561633e-15
9.01580010e-16 2.50764559e-05
-1.91018397e-06 -1.67478292e-08
9.07476614e-10]
Cl 1
[9.31739369e-21 -1.55916136e-12 -2.21353777e-13
4.65238697e-11 -1.71477490e-10 -1.68678861e-15
8.26787007e-16 2.31873222e-05
-2.12589148e-06 -1.61519061e-08 9.98214034e-10]
Cl 2
[-9.80102287e-21 -1.43191593e-12
-2.36528448e-13 4.17994980e-11
-1.64640303e-10 -1.80561633e-15
9.01580010e-16 2.50764559e-05
-1.91018397e-06 -1.67478292e-08
9.07476614e-10]
Cl 3
[-9.61056957e-21 -1.55215299e-12 -2.13631228e-13
5.06815310e-11
-1.72774675e-10 -1.83669770e-15 8.01392264e-16
2.45434626e-05
-1.98477743e-06 -1.70221284e-08
9.98214034e-10]

Diagram 2 -


Initial Population
Selected Parents
[-1.02475200e-20 -1.45799022e-12 -2.28980078e-13 4.62010753e-11 -1.71402395e-10 -2.01826673e-15 8.52944060e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.98214034e-10]
27.1%
316548237852.3015
[-8.55339467e-21 -1.44021970e-12 -2.35538180e-13 4.21078007e-11 -1.49318295e-10 -1.79786188e-15 8.92315689e-16 2.49531750e-05 -2.10006800e-06 -1.68532479e-08 9.94550974e-10]
19.2%
611268754959.5098
[-8.55339467e-21 -1.44021970e-12 -2.35538180e-13 4.21078007e-11 -1.49318295e-10 -1.79786188e-15 8.92315689e-16 2.49531750e-05 -2.10006800e-06 -1.68532479e-08 9.94550974e-10]
[9.04224241e-21 -1.47825304e-12 -2.08476850e-13 4.41339994e-11 -1.82347156e-10 -1.83669770e-15 8.05208489e-16 2.22232297e-05 -1.91982848e-06 -1.53633810e-08 9.45805688e-10]
18.05%
654251278740.2003
[9.04224241e-21 -1.47825304e-12 -2.08476850e-13 4.41339994e-11 -1.82347156e-10 -1.83669770e-15 8.05208489e-16 2.22232297e-05 -1.91982848e-06 -1.53633810e-08 9.45805688e-10]
[-9.80102287e-21 -1.43191593e-12 -2.36528448e-13 4.17994980e-11 -1.64640303e-10 -1.80561633e-15 9.01580010e-16 2.50764559e-05 -1.91018397e-06 -1.67478292e-08 9.07476614e-10]
17.5%
672838617336.4539
[9.31739369e-21 -1.55916136e-12 -2.21353777e-13 4.65238697e-11 -1.71477490e-10 -1.68678861e-15 8.26787007e-16 2.31873222e-05 -2.12589148e-06 -1.61519061e-08 9.98214034e-10]
9.4%
973203073095.0851
[9.39351493e-21 -1.40236616e-12 -2.09356992e-13 4.78179067e-11 -1.84008352e-10 -1.79228217e-15 8.52944060e-16 2.35235897e-05 -2.04435150e-06 -1.44526273e-08 9.25162125e-10]
6.7%
1073696003900.96
[9.39351493e-21 -1.40236616e-12 -2.09356992e-13 4.78179067e-11 -1.84008352e-10 -1.79228217e-15 8.52944060e-16 2.35235897e-05 -2.04435150e-06 -1.44526273e-08 9.25162125e-10] 1073696003900.96
[8.06976549e-21 -1.55085759e-12 -2.25724531e-13 4.42439574e-11 -1.71031309e-10 -1.63715616e-15 8.31667844e-16 2.22148277e-05 -2.13208630e-06 -1.46830648e-08 1.02710013e-09]
1.7%
1260186907517.092
[-9.61056957e-21 -1.55215299e-12 -2.13631228e-13 5.06815310e-11 -1.72774675e-10 -1.83669770e-15 8.01392264e-16 2.45434626e-05 -1.98477743e-06 -1.70221284e-08 9.98214034e-10]
0%
1325717782967.6301

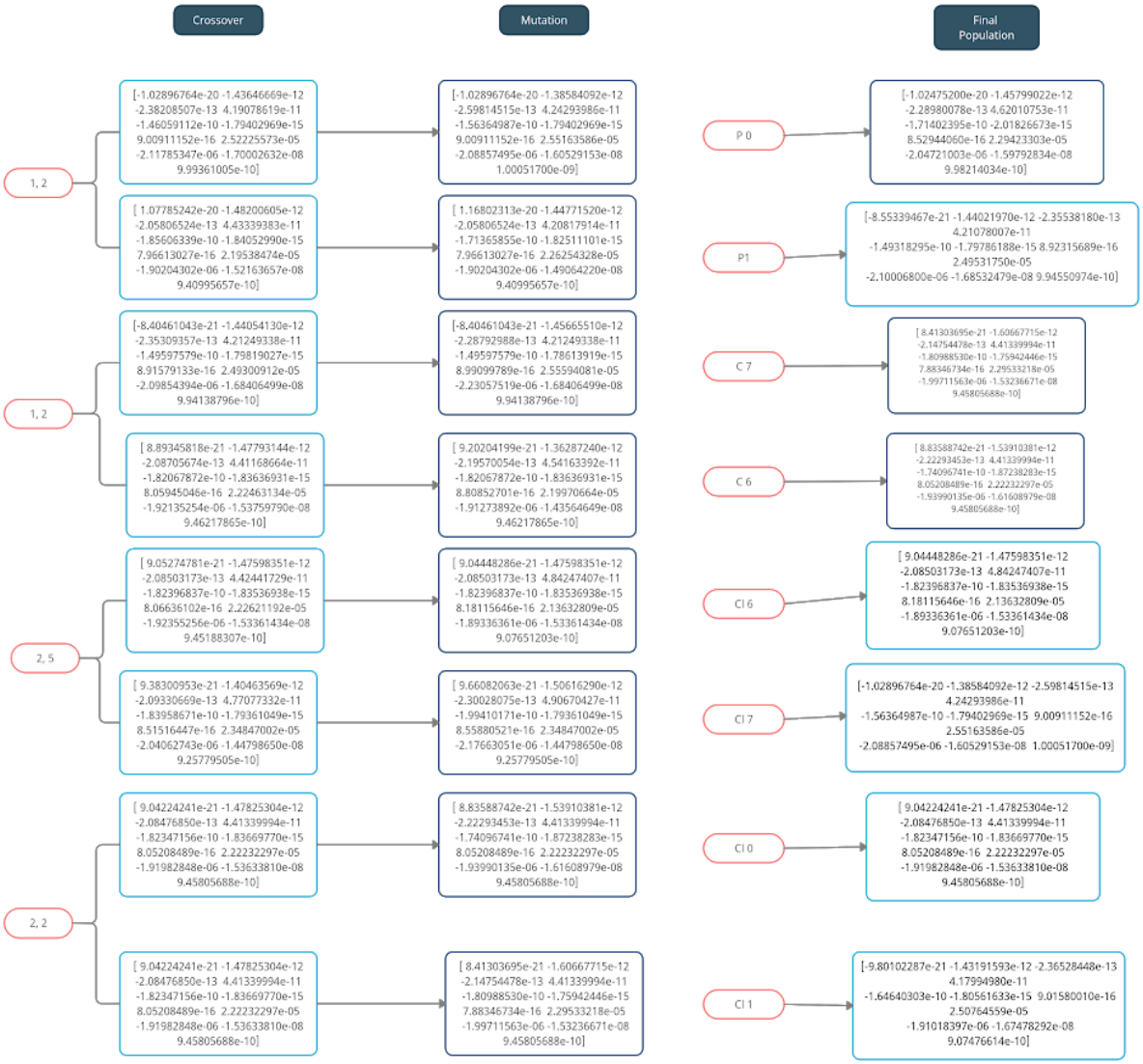
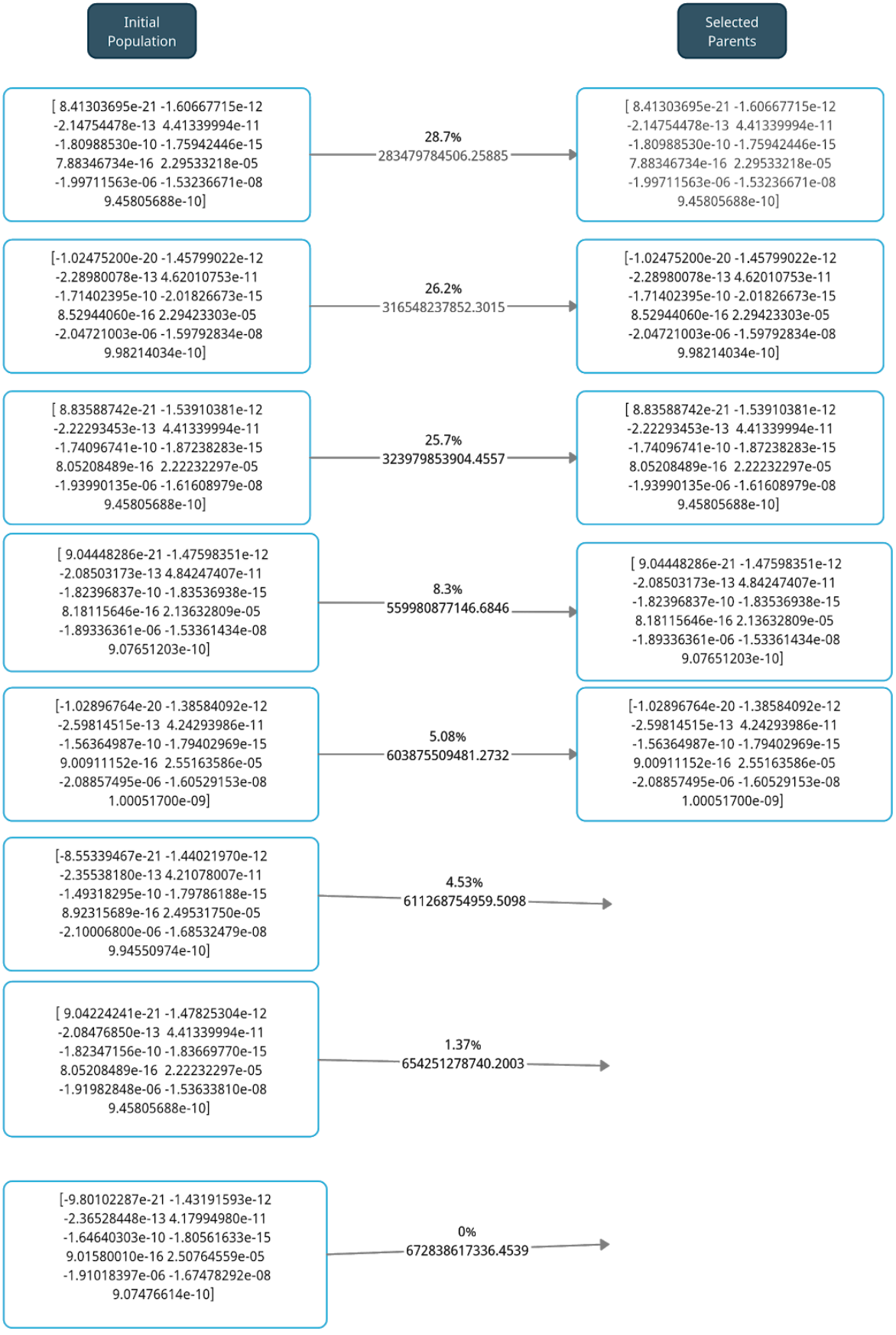
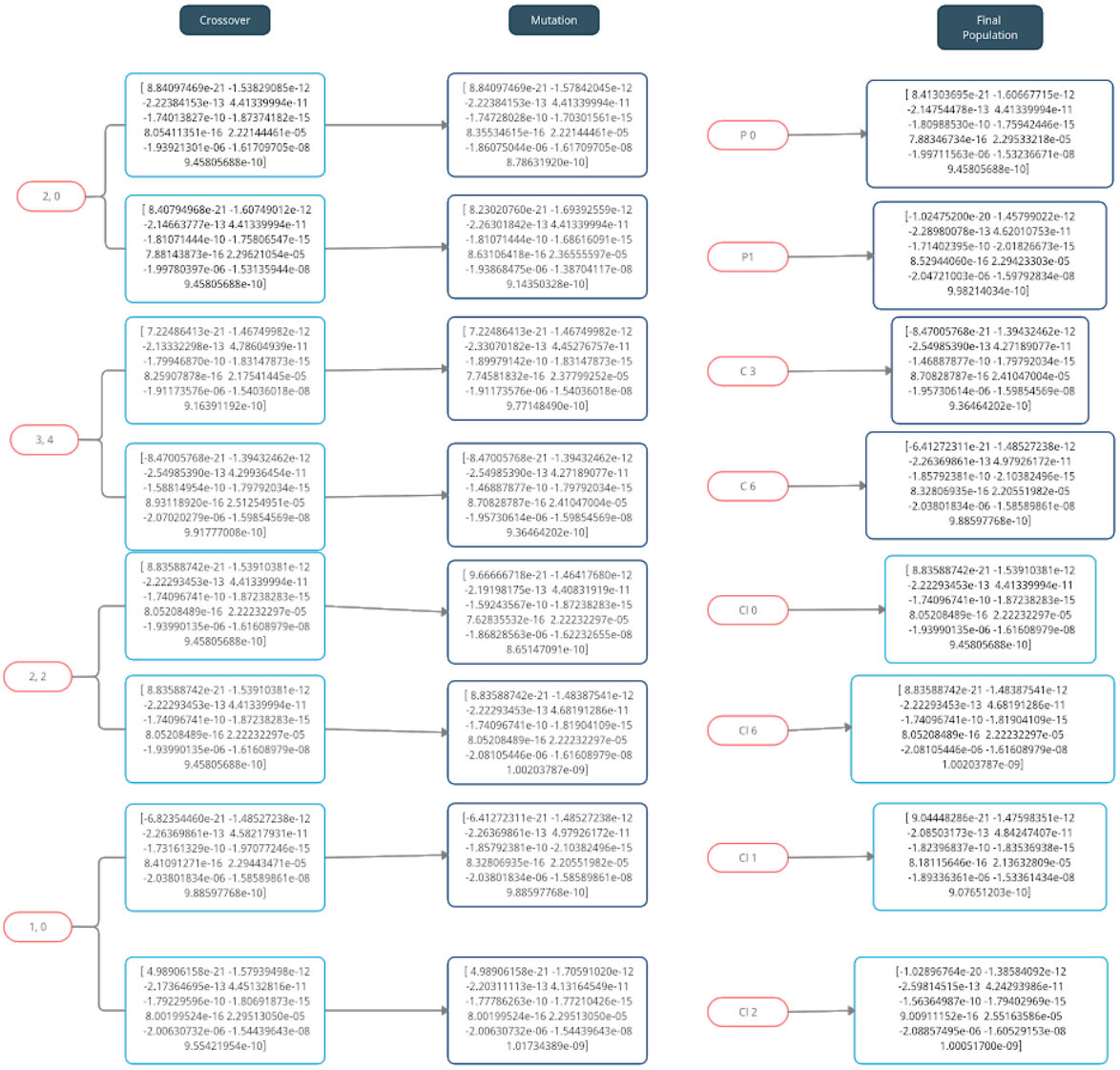

Crossover
Mutation
Final Population
1, 2
[-1.02896764e-20 -1.43646669e-12 -2.38208507e-13 4.19078619e-11 -1.46059112e-10 -1.79402969e-15 9.00911152e-16 2.52225573e-05 -2.11785347e-06 -1.70002632e-08 9.99361005e-10]
[-1.02896764e-20 -1.38584092e-12 -2.59814515e-13 4.24293986e-11 -1.56364987e-10 -1.79402969e-15 9.00911152e-16 2.55163586e-05 -2.08857495e-06 -1.60529153e-08 1.00051700e-09]
[1.07785242e-20 -1.48200605e-12 -2.05806524e-13 4.43339383e-11 -1.85606339e-10 -1.84052990e-15 7.96613027e-16 2.19538474e-05 -1.90204302e-06 -1.52163657e-08 9.40995657e-10]
[1.16802313e-20 -1.44771520e-12 -2.05806524e-13 4.20817914e-11 -1.71365855e-10 -1.82511101e-15 7.96613027e-16 2.26254328e-05 -1.90204302e-06 -1.49064220e-08 9.40995657e-10]
1, 2
[-8.40461043e-21 -1.44054130e-12 -2.35309357e-13 4.21249338e-11 -1.49597579e-10 -1.79819027e-15 8.91579133e-16 2.49300912e-05 -2.09854394e-06 -1.68406499e-08 9.94138796e-10]
[-8.40461043e-21 -1.45665510e-12 -2.28792988e-13 4.21249338e-11 -1.49597579e-10 -1.78613919e-15 8.99099789e-16 2.55594081e-05 -2.23057519e-06 -1.68406499e-08 9.94138796e-10]
[8.89345818e-21 -1.47793144e-12 -2.08705674e-13 4.41168664e-11 -1.82067872e-10 -1.83636931e-15 8.05945046e-16 2.22463134e-05 -1.92135254e-06 -1.53759790e-08 9.46217865e-10]
[9.20204199e-21 -1.36287240e-12 -2.19570054e-13 4.54163392e-11 -1.82067872e-10 -1.83639931e-15 8.80852701e-16 2.19970664e-05 -1.91273892e-06 -1.43564649e-08 9.46217865e-10]
2, 5
[9.05274781e-21 -1.47598351e-12 -2.08503173e-13 4.42441729e-11 -1.82396837e-10 -1.83536938e-15 8.06636102e-16 2.22621192e-05 -1.92355256e-06 -1.53361434e-08 9.45188307e-10]
[9.04448286e-21 -1.47598351e-12 -2.08503173e-13 4.84247407e-11 -1.82396837e-10 -1.83536938e-15 8.18115646e-16 2.13632809e-05 -1.89336361e-06 -1.53361434e-08 9.07651203e-10]
[9.38300953e-21 -1.40463569e-12 -2.09330669e-13 4.77077332e-11 -1.83958671e-10 -1.79361049e-15 8.51516447e-16 2.34847002e-05 -2.04062743e-06 -1.44798650e-08 9.25779505e-10]
[9.66082063e-21 -1.50616290e-12 -2.30028075e-13 4.90670427e-11 -1.99410171e-10 -1.79361049e-15 8.55880521e-16 2.34847002e-05 -2.17663051e-06 -1.44798650e-08 9.25779505e-10]
2, 2
[9.04224241e-21 -1.47825304e-12 -2.08476850e-13 4.41339994e-11 -1.82347156e-10 -1.83669770e-15 8.05208489e-16 2.22232297e-05 -1.91982848e-06 -1.53633810e-08 9.45805688e-10]
[8.83588742e-21 -1.53910381e-12 -2.22293453e-13 4.41339994e-11 -1.74096741e-10 -1.87238283e-15 8.05208489e-16 2.22232297e-05 -1.93990135e-06 -1.61608979e-08 9.45805688e-10]
[9.04224241e-21 -1.47825304e-12 -2.08476850e-13 4.41339994e-11 -1.82347156e-10 -1.83669770e-15 8.05208489e-16 2.22232297e-05 -1.91982848e-06 -1.53633810e-08 9.45805688e-10]
[8.41303695e-21 -1.60667715e-12 -2.14754478e-13 4.41339994e-11 -1.80988530e-10 -1.75942446e-15 7.88346734e-16 2.29533218e-05 -1.99711563e-06 -1.53236671e-08 9.45805688e-10]
P 0
[-1.02475200e-20 -1.45799022e-12 -2.28980078e-13 4.62010753e-11 -1.71402395e-10 -2.01826673e-15 8.52944060e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.98214034e-10]
P1
[-8.55339467e-21 -1.44021970e-12 -2.35538180e-13 4.21078007e-11 -1.49318295e-10 -1.79786188e-15 8.92315689e-16 2.49531750e-05 -2.10006800e-06 -1.68532479e-08 9.94550974e-10]
C 7
[8.41303695e-21 -1.60667715e-12 -2.14754478e-13 4.41339994e-11 -1.80988530e-10 -1.75942446e-15 7.88346734e-16 2.29533218e-05 -1.99711563e-06 -1.53236671e-08 9.45805688e-10]
C 6
[8.83588742e-21 -1.53910381e-12 -2.22293453e-13 4.41339994e-11 -1.74096741e-10 -1.87238283e-15 8.05208489e-16 2.22232297e-05 -1.93990135e-06 -1.61608979e-08 9.45805688e-10]
Cl 6
[9.04448286e-21 -1.47598351e-12 -2.08503173e-13 4.84247407e-11 -1.82396837e-10 -1.83536938e-15 8.18115646e-16 2.13632809e-05 -1.89336361e-06 -1.53361434e-08 9.07651203e-10]
Cl 7
[-1.02896764e-20 -1.38584092e-12 -2.59814515e-13 4.24293986e-11 -1.56364987e-10 -1.79402969e-15 9.00911152e-16 2.55163586e-05 -2.08857495e-06 -1.60529153e-08 1.00051700e-09]
Cl 0
[9.04224241e-21 -1.47825304e-12 -2.08476850e-13 4.41339994e-11 -1.82347156e-10 -1.83669770e-15 8.05208489e-16 2.22232297e-05 -1.91982848e-06 -1.53633810e-08 9.45805688e-10]
Cl 1
[-9.80102287e-21 -1.43191593e-12 -2.36528448e-13 4.17994980e-11 -1.64640303e-10 -1.80561633e-15 9.01580010e-16 2.50764559e-05 -1.91018397e-06 -1.67478292e-08 9.07476614e-10]

Diagram 3 -


Initial Population
Selected Parents
[8.41303695e-21 -1.60667715e-12 -2.14754478e-13 4.41339994e-11 -1.80988530e-10 -1.75942446e-15 7.88346734e-16 2.29533218e-05 -1.99711563e-06 -1.53236671e-08 9.45805688e-10]
28.7%
283479784506.25885
[8.41303695e-21 -1.60667715e-12 -2.14754478e-13 4.41339994e-11 -1.80988530e-10 -1.75942446e-15 7.88346734e-16 2.29533218e-05 -1.99711563e-06 -1.53236671e-08 9.45805688e-10]
[-1.02475200e-20 -1.45799022e-12 -2.28980078e-13 4.62010753e-11 -1.71402395e-10 -2.01826673e-15 8.52944060e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.98214034e-10]
26.2%
316548237852.3015
[-1.02475200e-20 -1.45799022e-12 -2.28980078e-13 4.62010753e-11 -1.71402395e-10 -2.01826673e-15 8.52944060e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.98214034e-10]
[8.83588742e-21 -1.53910381e-12 -2.22293453e-13 4.41339994e-11 -1.74096741e-10 -1.87238283e-15 8.05208489e-16 2.22232297e-05 -1.93990135e-06 -1.61608979e-08 9.45805688e-10]
25.7%
323979853904.4557
[8.83588742e-21 -1.53910381e-12 -2.22293453e-13 4.41339994e-11 -1.74096741e-10 -1.87238283e-15 8.05208489e-16 2.22232297e-05 -1.93990135e-06 -1.61608979e-08 9.45805688e-10]
[9.04448286e-21 -1.47598351e-12 -2.08503173e-13 4.84247407e-11 -1.82396837e-10 -1.83536938e-15 8.18115646e-16 2.13632809e-05 -1.89336361e-06 -1.53361434e-08 9.07651203e-10]
8.3%
559980877146.6846
[9.04448286e-21 -1.47598351e-12 -2.08503173e-13 4.84247407e-11 -1.82396837e-10 -1.83536938e-15 8.18115646e-16 2.13632809e-05 -1.89336361e-06 -1.53361434e-08 9.07651203e-10]
[-1.02896764e-20 -1.38584092e-12 -2.59814515e-13 4.24293986e-11 -1.56364987e-10 -1.79402969e-15 9.00911152e-16 2.55163586e-05 -2.08857495e-06 -1.60529153e-08 1.00051700e-09]
5.08%
603875509481.2732
[-1.02896764e-20 -1.38584092e-12 -2.59814515e-13 4.24293986e-11 -1.56364987e-10 -1.79402969e-15 9.00911152e-16 2.55163586e-05 -2.08857495e-06 -1.60529153e-08 1.00051700e-09]
[-8.55339467e-21 -1.44021970e-12 -2.35538180e-13 4.21078007e-11 -1.49318295e-10 -1.79786188e-15 8.92315689e-16 2.49531750e-05 -2.10006800e-06 -1.68532479e-08 9.94550974e-10]
4.53%
611268754959.5098
[9.04224241e-21 -1.47825304e-12 -2.08476850e-13 4.41339994e-11 -1.82347156e-10 -1.83669770e-15 8.05208489e-16 2.22232297e-05 -1.91982848e-06 -1.53633810e-08 9.45805688e-10]
1.37%
654251278740.2003
[-9.80102287e-21 -1.43191593e-12 -2.36528448e-13 4.17994980e-11 -1.64640303e-10 -1.80561633e-15 9.01580010e-16 2.50764559e-05 -1.91018397e-06 -1.67478292e-08 9.07476614e-10]
0%
672838617336.4539



Fitness Function:

The value of the fitness function is the determining factor for the selection of a population element in the next generation as well as the selection of parent elements for crossover. Let t denote the training error and v denote the validation error.

Initially, we started with a fitness function equal to t+v, to get the best vectors that have lower error values. But the vectors were observed to have a huge difference between training error value and validation error value. Thus, to minimize this gap, we used a fitness function with a value (t + v + |t-v|), where equal importance is given to the difference between the training and validation error value and the total error (sum of training and validation errors). Finally, when the sum of errors was low enough, to decrease the difference between t and v, and to prevent overfitting, we used the fitness function as |t-v|.

Crossover function:

The Russian roulette method provides a way to choose good quality parents (i.e. the parents with less error value) with more probability for crossover.
First, we take the negative of the fitness function as we are trying to minimize the fitness function value but the Russian roulette is defined for a maximizing problem. Then we normalize the fit function value such that the value of the fitness function ranges from 0 to 1.
Now we define the probability of selection of an element for crossover based on the fitness function value.
The probability of selection of an element as a parent at index i is the cumulative sum of the ratios given by (fitness function value) / (sum of fitness function values of all the elements).

For selecting the parent elements, 2 parents from the parent population are chosen by generating random numbers and comparing them with the calculated probabilities of the elements.

For the generation of the child population, the crossover technique used is Simulated Binary crossover. This method generates two children which satisfy the equation -

$$\frac{x_1^{\text{new}} + x_2^{\text{new}}}{2} = \frac{x_1 + x_2}{2}$$

For the crossover we calculate beta, for which we use the distribution index eta_c and u is a random number generated between 0 and 1. The value of beta is calculated using -

$$\beta = \begin{cases} (2u)^{\frac{1}{\eta_c+1}}, & \text{if } u \le 0.5 \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{\eta_c+1}}, & \text{otherwise} \end{cases}$$

From the value of beta and the parent vectors x1 and x2, the children can be generated by using the equations -

$$x_1^{\text{new}} = 0.5[(1+\beta)x_1 + (1-\beta)x_2]$$
$$x_2^{\text{new}} = 0.5[(1-\beta)x_1 + (1+\beta)x_2]$$

Some of the features of the Simulated Binary crossover are -

- Generated children vectors have the same distance from the average value of parents vectors involved in crossover.
- Each point of the vector has the same probability to be selected as a crossover point.
- The crossover in the lower bit results in small variations in the vector
- The generated Children vectors are more likely to be near the parent vectors.

Mutation function:

The mutation function involves -

- Considering a vector ("vec") of the population obtained after crossover, for every element "vec[i]" a random number is generated. If this random number is less than mutation probability, then the next step is carried out.
- The mutation range is set as -range_val to range_val where range_val is passed from the main function. Different values for range_val were tried and finally range_val=0.1 was fixed.
- A random number (denoted by num) lying within the mutation range is generated.
- The modified element is given by vec[i] * (1 + num).

After trying out various values for the mutation probability, we decided to settle on the value of mutation probability as 0.5 for creating the child population. Further details about mutation probability are mentioned in the below section.

Hyperparameters:

1. Population size:
   We started from the population size of 10 and increased it to 20 later. But considering the restricted number of requests that were allowed, we decreased the population size to 16 so that we can have a variety in the population and also be able to generate a good number of generations in a day.

2. Number of elements passed to the new generation:
   The variable n denotes the number of top elements from the parent and child population that will be included in the newly formed generation. This is done to ensure that the best vectors appearing in the parent and the child population will also appear in the new generation. After trying several values we finally decided to use n = 4

3. Mutation probability:
   For creating the initial population, the mutation probability is taken to be 0.9 to create a diverse population from the initial weight vector. The initial weight vector is also included in the initial population after mutating with a probability of 0.4

In the case of the rest of the generations, for setting the value of mutation probability, we tried different values between 0.3 and 0.9. We observed that keeping the mutation probability high caused a modification in the vector such that it deviated away from the best answer we got. And keeping the mutation probability low, the vector did not modify much and so the result was also not improving. So finally we settled for mutation probability = 0.5 as this seemed to work well most of the time.

4. Mutation range:
   Setting up the mutation range as -range_val to range_val where range_val is passed from the main function. We tried different values for range_val like 0.01, 0.05 and finally decided to use range_val=0.1
   A random number (denoted by num) lying within the mutation range is generated. The mutated element is equal to vec[i] * (1 + num).

5. Distribution index:
   The variable η_c (eta_c in the code) denotes the distribution index. It is a non-negative real parameter. A large value of η_c gives a higher probability for creating near parent solutions and a small value of η_c allows distant solutions to be selected as children solutions. We tried a range of values for η_c between 3 to 5, and finally decided to use the value as $\eta_c = 3$

Statistical Data:

It was observed that the number of iterations required for the algorithm to converge ranged from 80 - 100.

Heuristics applied:

1. Choosing the parent indices and crossover technique:
   Initially, we were using a random selection of parents from top k=6 parents, and a crossover function where a pivot index was chosen randomly and all the elements appearing after the pivot index in the chosen parent elements are swapped. That is if p1 and p2 are chosen parents and the pivot is at index i, then all the elements from index i+1 onwards in p1 will be placed in p2 and elements after index i in p2 will be placed in p1 (single point crossover).
   Finally, we decided to use Russian roulette for parent selection to ensure that vectors with less error value have more probability of being chosen for crossover rather than choosing the parents randomly. Further we decided to use simulated Binary crossover for generating children in order to ensure the final children generated didn't vary too much or too little from the original parents.
2. Fitness function:
   We had started with the fitness function as t+v. This fitness function resulted in vectors that had lower values of the above summation.

But we were getting a quite higher value of validation error as compared to training error. Thus to reduce the difference between the training error and the validation error we decided to define the fitness function as t + v + |t-v|, giving equal importance to reducing the gap between the Training and the Validation error.
While running code and comparing the values for different iterations, we tried various expressions of fitness function like

- t+v
- |t-v|
- t+v+|t-v|
- t+v + 2*|t-v|
- t+v + 1.5*|t-v|
- 2*(t+v) + |t-v|

3. Mutation function:
   For the mutation of a vector, initially, we were adding a random number between -0.01 to 0.01 to some of the elements in the vector based on the mutation probability. (If a random number generated between 0 to 1 is less than the mutation probability, then the value of the element in the vector is modified). We realised that all the elements were being added to the same number(of order 10^{-2} or less) and this number was much larger than most of the vector elements. This resulted in huge deviations from the original value especially for the really small numbers (numbers in the order of 10^{-11} and lesser).
   Thus it was needed to mutate the number such that the modification or the scaling should not cause a huge deviation from the original vectors.

<u>Value of Training and Validation error:</u>
The least values of errors we were able to achieve -

- Train error value = 60846388949.14433
- Validation error value = 60106811750.99192

These values were obtained when we were trying to minimize both the total error and the difference between the training and validation errors i.e. (when the fitness function was t+v+|t-v|). In further iterations, we focussed on decreasing the difference between the two errors and preventing overfitting at the same time. Thus we changed the fitness function to |t-v| and this gave us the following errors:

- Train error value = 72966509358.28659
- Validation error value = 72941599287.8962

<u>Why our algorithm would perform well on unseen data</u>:
The Genetic algorithm involves a process where after each iteration, we pick the best vectors (vectors with the least error) till now. These “parent” vectors are selected using the russian roulette method which assures that the best vectors(vectors with lower error values) will have a higher probability of being

chosen as parents. The chosen pair of vectors are crossed to generate new vectors. This crossing is done using Simulated binary crossover algorithm. Larger value of the distribution index implies lesser deviation from parents and vice versa. The distribution index is varied several times throughout the procedure to obtain the best children. The children produced have the same distance from the average value of the parents These newly generated vectors are mutated to obtain a slight variation of the vectors obtained after crossing. And finally the best vectors from the original parent generation and the child generation are picked to form the new generation of vectors.

In short, in this algorithm we are always picking vectors that yield less error values, which results in decrement of the error value after nearly every iteration. Thus finally we will obtain a generation of vectors for which the error value is quite low, and the best vectors are obtained irrespective of what the dataset is. Thus, the genetic algorithm performs well even on unseen datasets.