REAL CODED GENETIC ALGORITHM

Mohssen Elshaar – g202309590

Contents

Table of Figures

Problem description & Report Flow

Objective Function

$$f(x_1, x_2, x_3) = x_1^2 + 2x_2^2 + 3x_3^2 + x_1x_2 + x_2x_3 - 8x_1 - 16x_2 - 32x_3 + 110,$$

$$Where, 0 \leq x_1, x_2, x_3 < 10$$

Report Flow

First, the report tackles the sensitivity analysis of the Genetic Algorithm hyperparameters for this specific objective function. This is done to specify the suitable range of each parameter to solve the problem with high accuracy. Secondly, a feature in the algorithm - I named population refreshment – is tested to see if it increases the reliability of the single run of the algorithm. If this test showed positive effect much computational cost could be saved when solving this and other problems. Finally, the 10 combinations requested in the Assignment are run with suitable parameters ranges, and with the refreshment feature after it showed a high performance boost to the algorithm.

Part I: Parameter Sensitivity Analysis

This part is a sensitivity analysis that investigates the Genetic Algorithm different hyperparameters. Those are: (1) "Stronger to be selected probability" in the Tournament Selection, (2) Cross-Over Probability, (3) Mutation Probability, and (4) Elite Solutions Percentage of the solution population. The Discussion of this part comes after the sensitivity reports.

Procedure

The Algorithm parameters are described as follows: (a) A population size of 20 solutions, each solution composed of 3 variables, and each variable is bounded between {0,10}. (b) The tournament selection was chosen as the selection method, Blend Cross-Over as the cross-over method, and random mutation is allowed. (c) The algorithm should stop if the number of generations reached 100, or if the change of the difference between the Global Optimum and the Ongoing-Generation Local optimum became negligible (less that 10^{-6}). Every analysis was repeated 10 times to get an insight of the best performance of each hyperparameter value. For Example, the Cross-Over probability is chosen from a pole of uniform distribution $\in$ [0.5, 0.9]. The algorithm was run for each value of Cross-Over probability 10 times, then the best run was considered for further analysis. These procedures are followed for each hyperparameter. Keeping in mind that this is a sensitivity analysis, for each hyperparameter changing, all other parameters are kept constant. Figure 1 shows the 10 runs done for each parameter sample value.

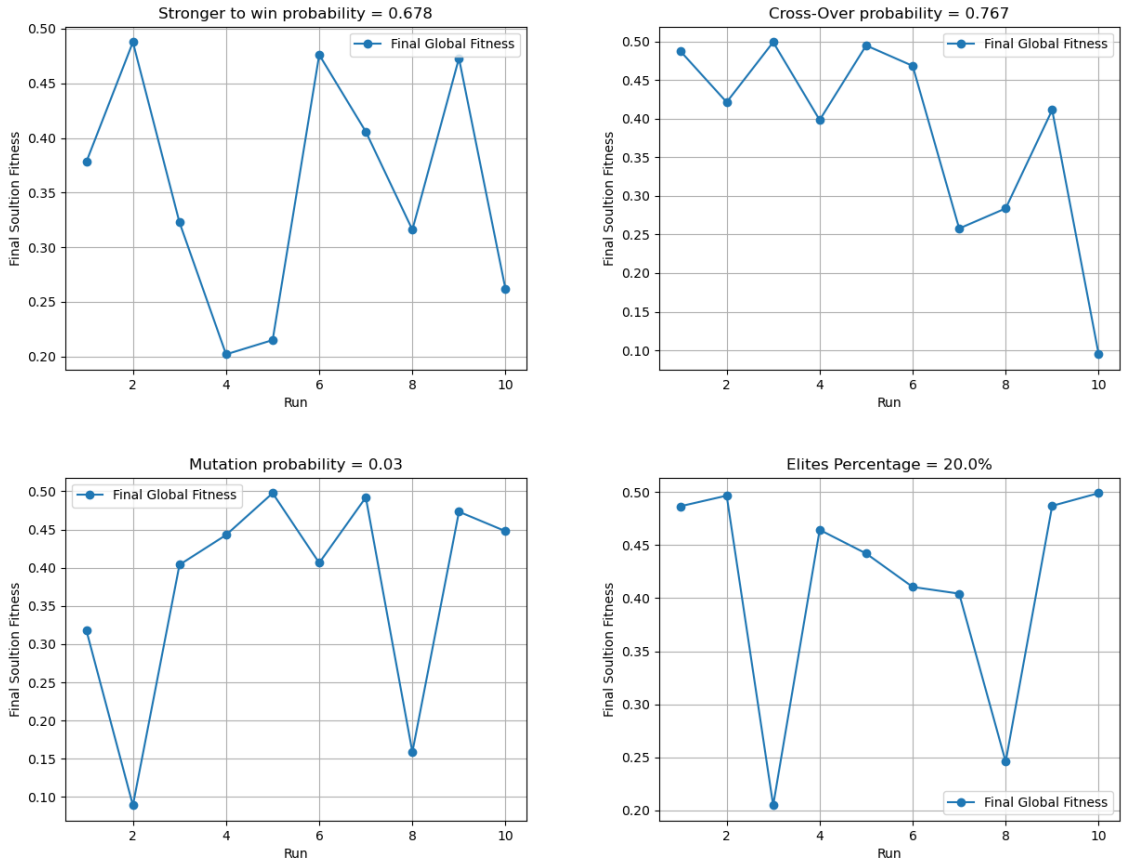


Figure 1 (plot 1 – 4: Repetitive Runs)

Stronger to be selected (STBS) probability

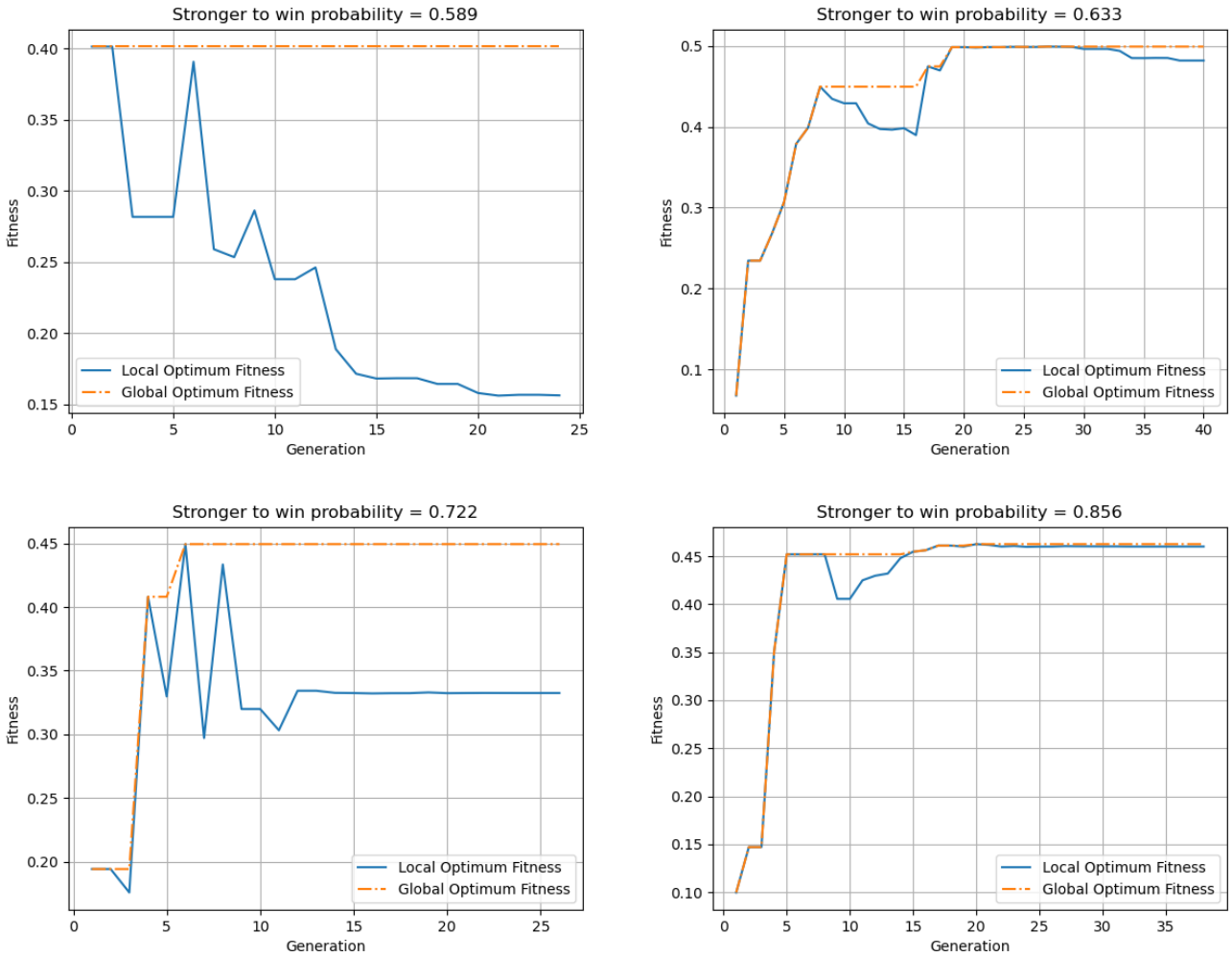
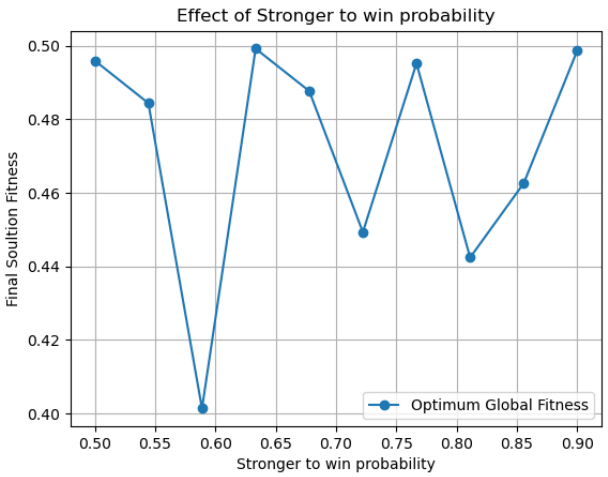


Figure 2 (plots 5 – 8: Stronger to Win Optimum Fitness vs Generations)



STBS Probability	x_1	x_2	x_3	*Fitness*
0.5000	2.9080	2.0401	5.0465	0.4959
0.5889	2.3357	2.3834	5.0235	0.4014
0.6778	3.1166	1.9772	5.1173	0.4877
0.7667	3.0313	2.0900	4.9773	0.4952
0.8556	3.1909	1.7173	5.1403	0.4626
0.9000	3.0043	1.9476	5.0087	0.4987

Figure 3 (plot 9 – table 1: Stronger to Win Effect)

Cross-Over probability

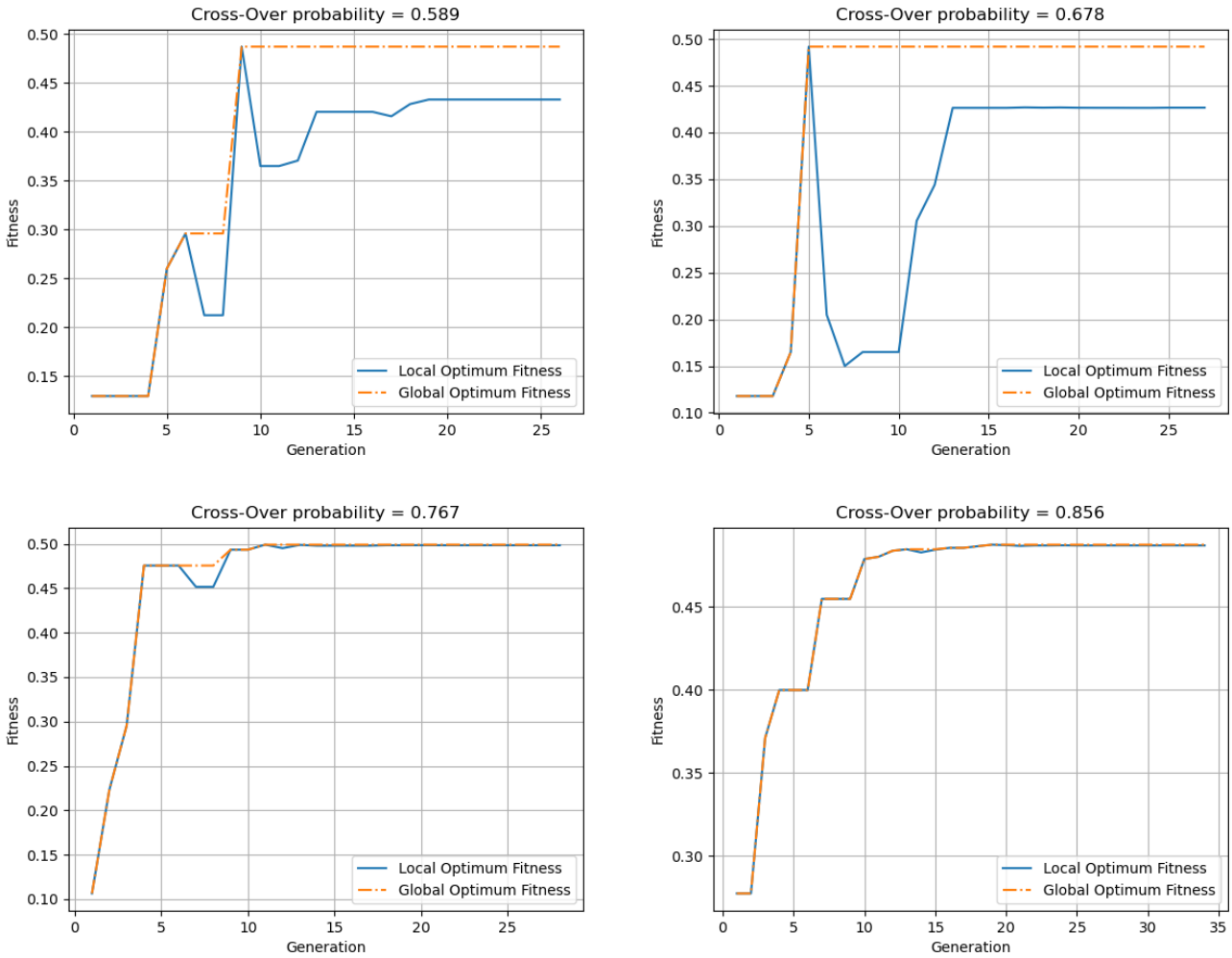
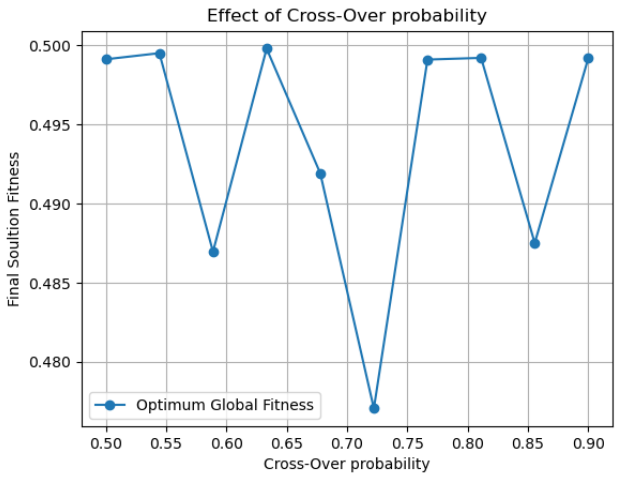


Figure 4 (plots 10 – 13: Cross-over Optimum Fitness vs Generations)



Cross-over Probability	x_1	x_2	x_3	*Fitness*
0.5000	2.9407	2.0303	4.9967	0.4991
0.5889	2.8987	1.9222	4.9239	0.4870
0.6778	3.1672	1.9842	4.9542	0.4919
0.7667	3.0281	2.0259	5.0114	0.4991
0.8556	2.9769	2.1519	5.0323	0.4875
0.9000	3.0146	2.0159	4.9703	0.4992

Figure 5 (plot 14 – table 2: Cross-Over Effect)

Mutation probability

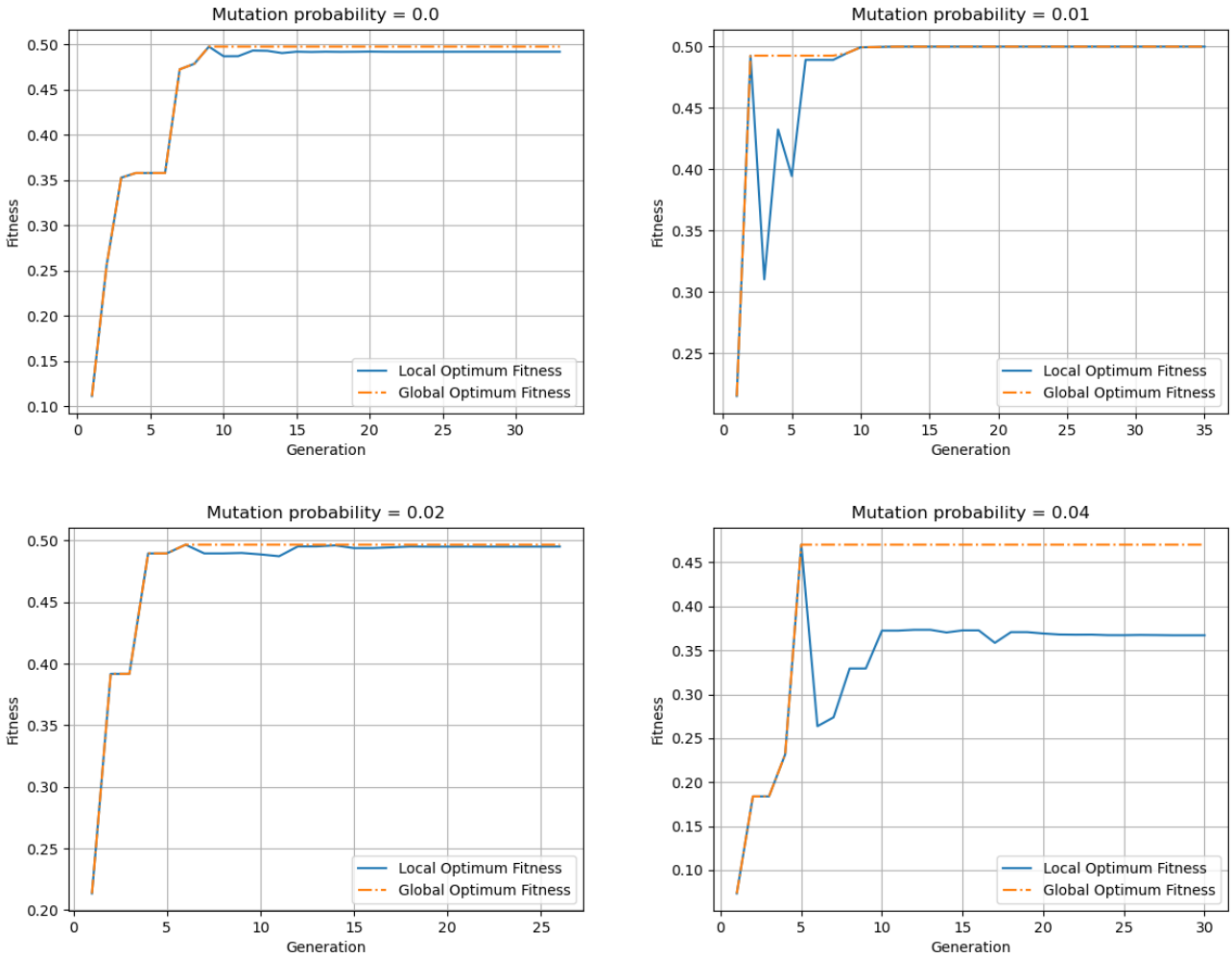
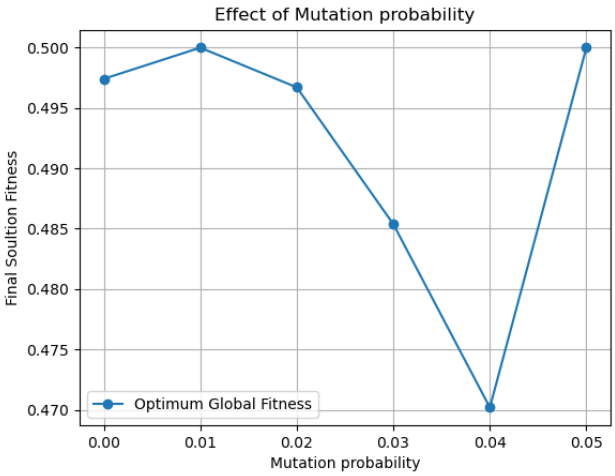


Figure 6 (plots 15 – 18: Mutation Optimum Fitness vs Generations)



Mutation Probability	x_1	x_2	x_3	*Fitness*
0.0000	3.0184	2.0108	4.9415	0.4974
0.0100	3.0000	2.0007	4.9988	0.5000
0.0200	3.1099	1.9945	5.0247	0.4967
0.0300	3.0850	2.1455	4.9710	0.4854
0.0400	3.3067	1.9741	5.1189	0.4702
0.0500	3.0029	1.9978	5.0010	0.5000

Figure 7 (plot 19 – table 3: Mutation Effect)

Elitism

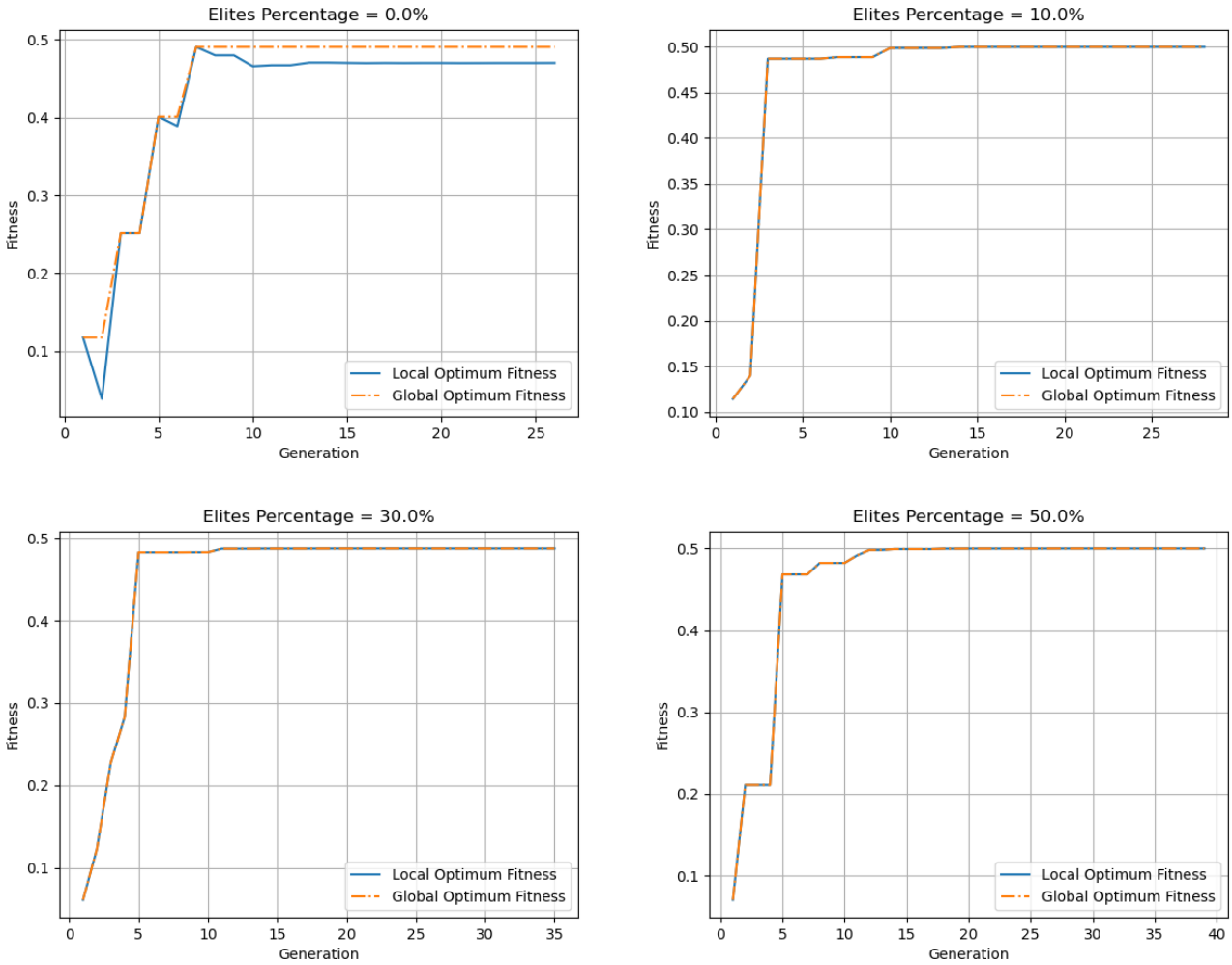
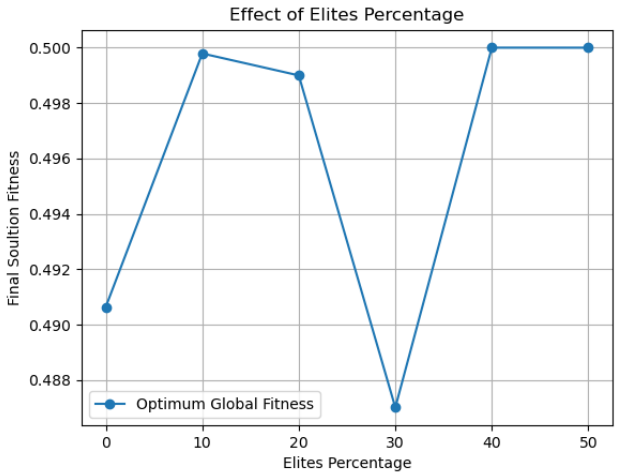


Figure 8 (plots 20 – 23: Elitism Optimum Fitness vs Generations)



No. of Elite solutions	x_1	x_2	x_3	*Fitness*
0	3.0744	1.9048	4.9296	0.4906
2	3.0170	1.9853	5.0138	0.4998
4	2.9448	2.0223	5.0167	0.4990
6	3.1160	1.8405	5.0832	0.4870
8	2.9997	2.0005	4.9989	0.5000
10	2.9972	2.0008	4.9991	0.5000

Figure 9 (plot 24 - table 4: Elitism Effect)

Discussion

It makes sense that the more probable a better solution is selected as a parent for future generations, the more the probability of having a favorable solution in the future increases becomes. However, there must be a percentage of bad solutions which succeed in having a fingerprint on the future generations of solutions as well, not to lose the diversity of the solutions, and consequently the continuity of the algorithm. Overall, the results in *Figure 2* and *Figure 3* show that increasing the "Stronger to be selected probability" – considerably – increases the overall performance of the Genetic Algorithm, and thus, provides a better final solution. Though, if the relatively stronger solutions are not strong when seen irrelative of the weaker solutions, increasing their survival probability narrows the search space to a region that maybe is not that good (*Figure 2: plot 7 & 8*).

When it comes to the cross-over probability, which resembles the probability of "mixing traits" of different solutions, it seems – from the results – that the higher this probability is the opportunities of having a very good over-all population increases. Nevertheless, when this probability is further increased, more than a certain random threshold, the solution population starts to get worse. This could be explained by the fact that a high cross-over probability means nothing but more "bad traits" are getting mixed with "good ones" to create future generations of solution. Although usually crossovers may lead to excellent traits, meaning better solutions, it could be concluded that an inconsiderably high cross-over probability will probably have a negative effect on the Genetic Algorithm performance. To have an insight of that, one should consider the analytical solution $\bar{x} = [3\ 2\ 5]$. Each of the six final optimum solutions in *Table 2* has some "good traits" or variables whose values are very close to the true ones, yet these solutions have a single variable or two that is not good enough. It is an advantage and a disadvantage at the same time, bad solutions may get better, and meanwhile, good solutions may get worse. It also worth noting that by increasing the cross-over probability along with a high "Stronger to be selected probability", the future generations will tend to lose their diversity (*Figure 4: plot 12 – 13*), which is not desirable in most cases.

Mutations are a means to unpredictable changes, and in similarity to nature, it has a very low probability. Mutations in the offspring or the future generations is like inducing a very simple, small, negligible randomness to the nearly systematic flow of the Genetic Algorithm. However small this change is, it would, surprisingly, cause a very noticeable positive change in the performance of the Genetic Algorithm. Keeping the mutation probability in its small, accepted range should always feed the algorithm with the randomness it naturally requires. *Figure 6* and *Figure 7* clearly shows the 2 side-effects of mutation; some mutations would lead to new induced randomness that widens the search region, deviating from the latest global optimum – Which is sometimes good for exploration – , while other mutations may hit the right target creating a "superior" solution, like those of fitness 0.5 (Analytical fitness) in *Table 3*.

Finally, elitism guarantees one thing; that the best solution of any upcoming generation is at least as good as the all-time or global best solution (*Figure 8: plots 21, 22 & 23 vs plot 20*). This consequently increases the chance that the generations get better as they advance. Increasing the elite solutions percentage in the whole population means more trust in the explored solutions, along with more mistrust in those solutions that are still unknown. This trust (percentage) should be considerate and dependent on the problem being solved. Generally speaking, a very high elites percentage makes the generations dependent on the initial population, which is totally random, and so this dependency could be favorable at one time and undesirable

at another. Genetic Algorithm flow should gradually untangle the dependency of latter generations on the preceding ones. Even if the fingerprint of the ancestors remains in the very last offspring, which is a must, the fingerprint of the direct parents should be plainer than those of the grandparents and so on reaching the initial population.

To sum up, the probability parameters, different selection, cross-over, and mutation methods and elitism are tackling the risk and benefits trade-off in randomness. The results in this report show the importance of striking the right balance between exploration and exploitation in a search space for optimal results. Isn't AI all about that?

Part II: Population Disruption & Algorithm Refreshment

In the first part, it was important to run the algorithm several times for a single hyper parameter value, just to make sure that the best run is probably close to the highest performance the algorithm could reach for that specific parameter value. Instead of doing so, I added a feature to the algorithm that if the Global Optimum isn't changed for more than 20% of the allowed number of generations, the ongoing solution population in that case should be randomized. This feature I would describe as "aggressive" and could be tuned for a partial randomization or maybe "induced mutations" in its better versions. In this part, the effectiveness of this population disruption (refreshment) on the overall performance of the algorithm; will it result in a better Final Global Optimum?

Procedure

1. 5 Random combinations of hyperparameters are produced to run with and without refreshment.

Combination	STBS	*Cross-Over*	*Mutation*
1	0.7031	0.8818	0.0271
2	0.8324	0.7994	0.0111
3	0.7868	0.8105	0.0396
4	0.8618	0.7533	0.0167
5	0.7834	0.7234	0.0390

Figure 10 (table 5: Refreshment test Combinations)

2. Each combination is run 10 times without refreshment, and another 10 times with refreshment. The average of the global optimum of 10 runs for each combination should be an indication of how reliable the algorithm output is. A lower average means that the algorithm's output isn't reliable enough, and repetitive runs are required. On the other hand, a higher average, that which is close to the analytical solution fitness guarantees more reliability.

Comparison

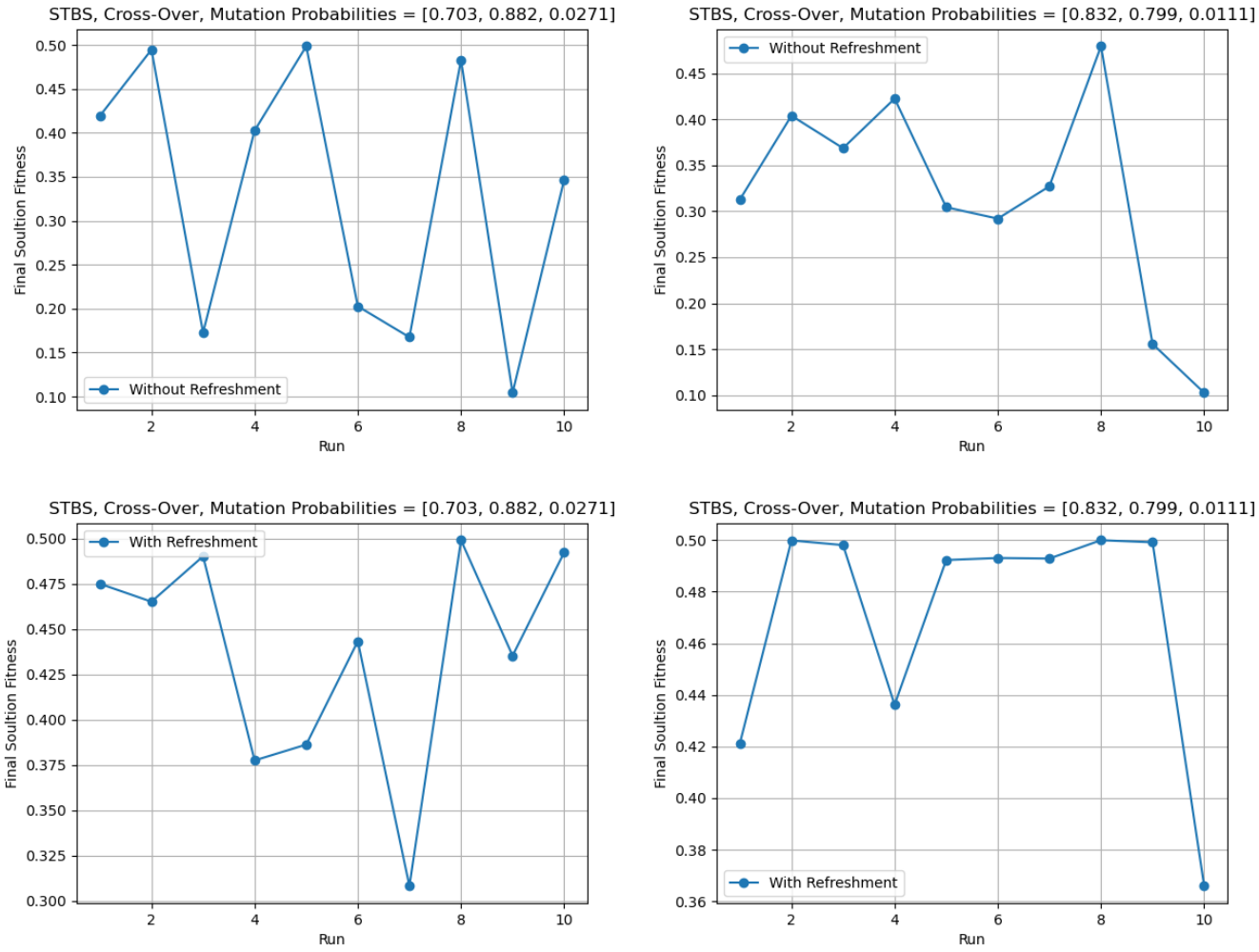


Figure 11 (plots: 25 – 28: Global Optimum Fitness vs Runs (With & Without Refreshments))

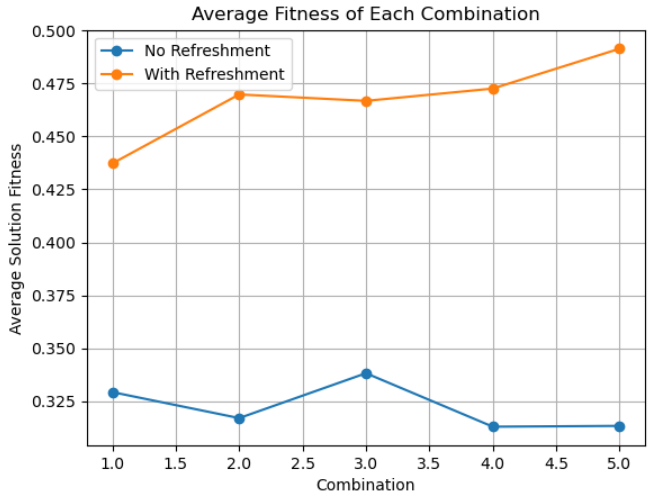


Figure 12 (plot 29: Average Fitness vs Combinations (With & Without Refreshments))

Discussion

According to *Figure 12*, the refreshment features enhanced the overall performance of the Genetic Algorithm, letting the final Global Optimum value to be more reliable. The minimum increase in fitness the refreshment feature provided was at the first combination, and it could be calculated:

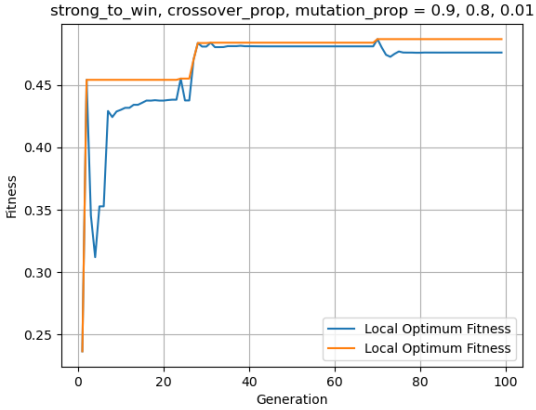
$$Increase\ Percent\ = \frac{0.4375 - 0.33}{0.4375} \times 100\ =\ 24.57\%$$

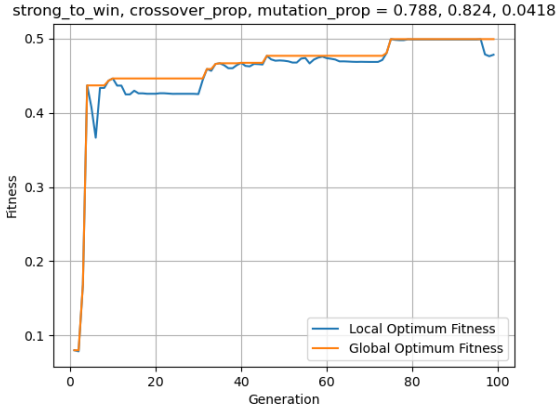
Knowing that the optimum fitness value is 0.5, the combinations accuracy with and without refreshment could be calculated:

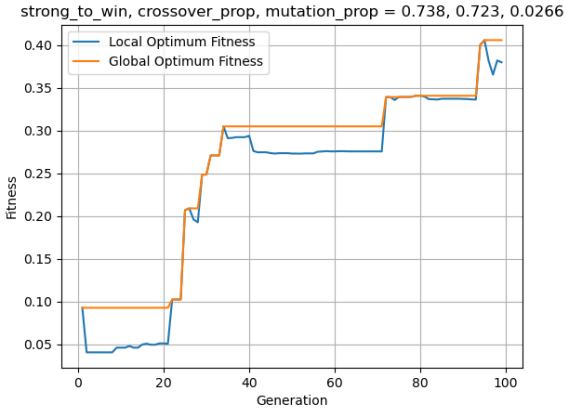
Combination	Accuracy With Refreshment	Accuracy *Without Refreshment*	*Accuracy Difference*
1	87.5 %	66 %	21.5 %
2	94 %	63 %	31 %
3	93 %	67.5 %	25.5 %
4	94.8 %	62.5 %	32.3 %
5	98 %	62.5 %	35.5 %
Average	**93.46 %**	**64.3 %**	**29.16 %**

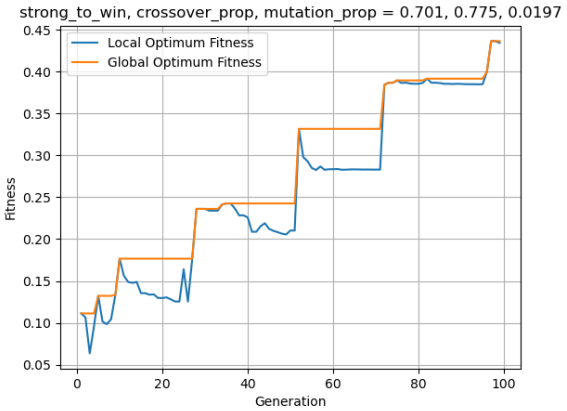
Figure 13 (table 6: Solution Accuracy)

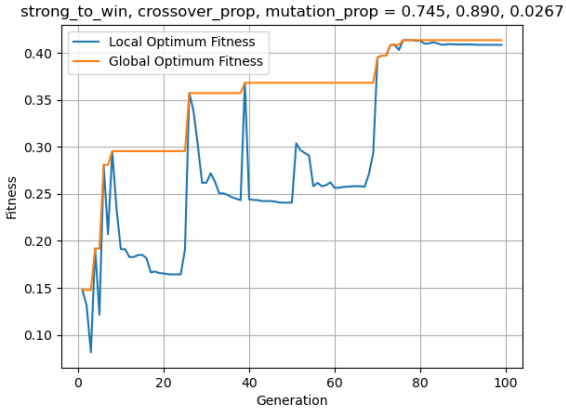
Part III: EE 556 Assignment 3 Requirements

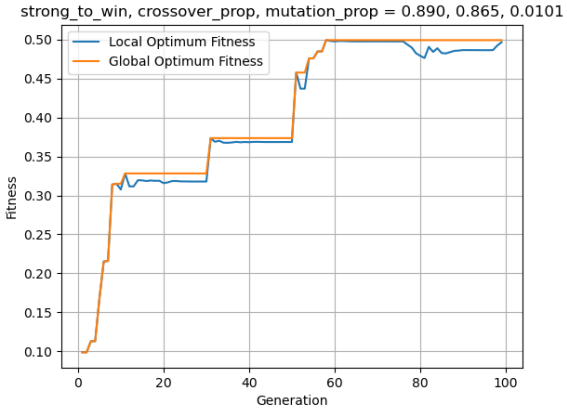












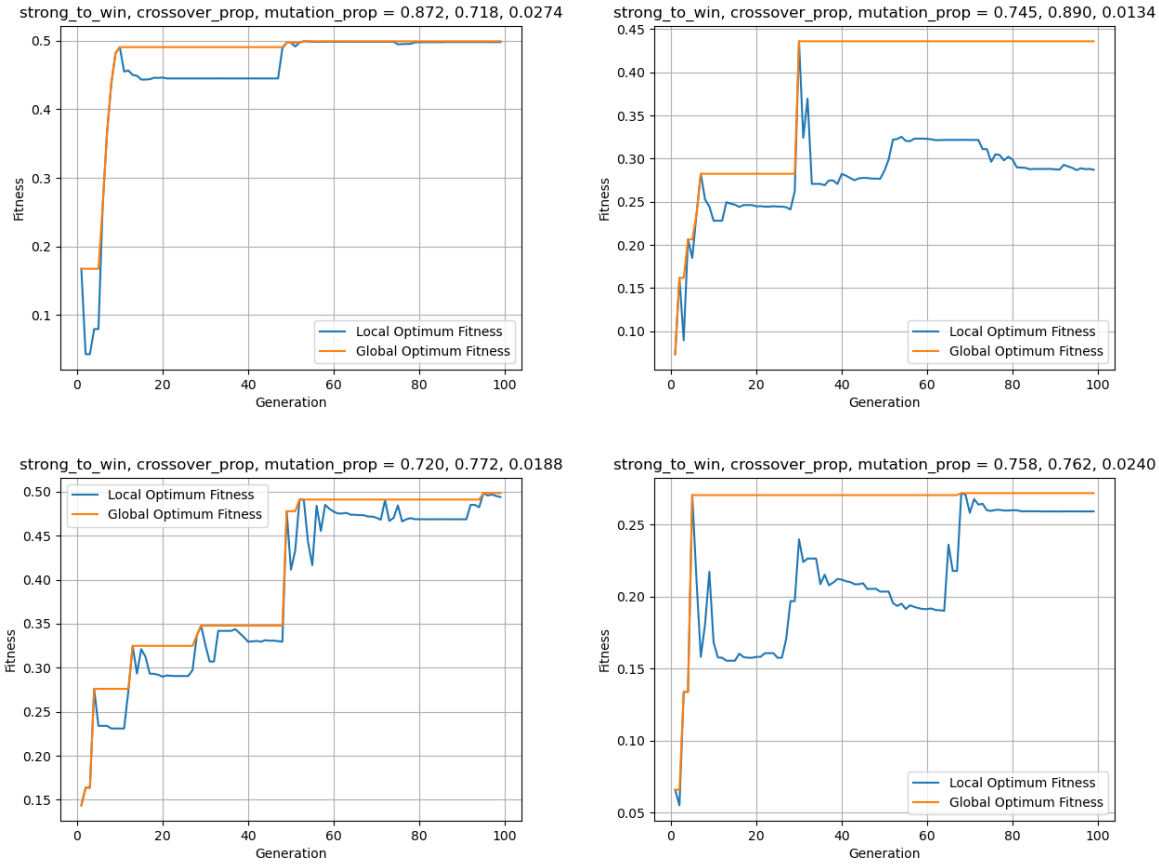


Figure 14 (plots 30 – 39: 10 Assignment Runs plots)

Run	*Stronger to Win Probability*	*Cross-over probability*	*Mutation Probability*	x_1	x_2	x_3	*Fitness*	*Error from Analytical*
1	0.9000	0.8000	0.0100	2.9728	2.0635	4.8636	0.4868	2.64 %
2	0.7879	0.8239	0.0418	3.0262	2.0239	5.0002	0.4994	0.12 %
3	0.7381	0.7234	0.0266	3.0287	2.4755	4.9914	0.4060	18.8 %
4	0.7015	0.7749	0.0197	2.7298	2.3224	4.7586	0.4365	12.7 %
5	0.7450	0.8895	0.0267	3.0845	2.4401	4.9031	0.4137	17.26 %
6	0.8895	0.8648	0.0101	3.0197	1.9920	5.0287	0.4994	0.12 %
7	0.8715	0.7185	0.0274	3.0581	1.9722	4.9889	0.4990	0.2 %
8	0.7451	0.8903	0.0134	3.1839	2.2539	4.7849	0.4360	12.8 %
9	0.7197	0.7716	0.0188	2.9265	1.9898	5.0130	0.4983	0.34 %
10	0.7583	0.7615	0.0240	3.0990	2.9047	4.9077	0.2719	45.62 %

Figure 15 (table 7: 10 Assignment Runs table)

Code and Future Work

The code was fully developed by the writer of this report and is flexible to use with any problem (just tuck your objective function in). The code allows for multiple selection, cross-over, elitism, and mutation methods. The code includes the population refreshment feature which I would describe as "aggressive" and could be tuned for a partial randomization or maybe "induced mutations" in its better versions. Also, the elite placement in the population could be further improved by choosing the worst solutions to be replaced rather than a random replacement.

Link to code on GitHub:
https://github.com/Hosnooo/Evolutionary-Optimization/blob/main/RC%20Genetic%20Algorithm.py