CIFO Project Report – Image Classification

MNIST Handwritten Digits

Iryna Savchuk - 20211310 - m20211310@novaims.unl.pt
Cátia Parrinha - 20201320 - m20201320@novaims.unl.pt

Group 17

https://github.com/iryna-savchuk/CI4O_project

1. Introduction

This project employs Genetic Algorithms (GAs) to optimize the weights of a single-layer neural network (NN) for image classification on the MNIST dataset. The MNIST dataset (a classic in the Machine Learning community) is a large dataset of handwritten digits assembled by the National Institute of Standards and Technology in the 1990s. This dataset contains 60,000 training images and 10,000 test images, all of the size 28x28 pixels and belonging to 10 classes.

The primary focus of this project is exploring and implementing GAs to solve a specific optimization problem, which is why we did not focus on Deep Learning methods much and have chosen a basic NN architecture to be optimized. In particular, the network under consideration contains only 3 layers: an input layer (consisting of 28x28 = 784 neurons); one hidden layer (consisting of 512 neurons); and an output layer (having 10 neurons in correspondence to 10 target labels).

In the project, several selection methods have been implemented as well as several mutation and crossover operators. We compare performance across multiple GA configurations and, in the end, choose three best-performing algorithms to run the final analysis and accomplish the project goal.

2. Implementation

To apply GAs to solve the MNIST image classification problem, we built our Python code based on the functions and classes that were developed during practical classes on Computational Intelligence for Optimization in the Nova IMS. We then had to adjust the representation of individuals, extend some methods and functions, and add several new genetic operators.

Representation: Every individual was encoded as a dictionary with 4 fields: ‘weights’ (a one-dimensional array of all weights and biases in the NN), ‘input’, ‘hidden’, and ‘output’ values (integers representing the number of neurons in the input, hidden, and output layers correspondingly; used to compile a fully-connected NN initialized with ‘weights’ to further calculate the Fitness of the individual). We had to adjust the constructor and other methods in the Individual and Population classes to support this representation.

Fitness: We designed a Fitness function for the current project to be a categorical accuracy estimated on the MNIST training dataset. The higher the accuracy, the better the model is in predicting labels of handwritten digits, so the purpose of GA in this case is to maximize the fitness of the individual solution.

Selection Algorithms: We used the two selection algorithms that were developed during the practical classes: Fitness Proportional Selection (FPS, or Roulette Wheel) and Tournament Selection. In the latter, we have set the default tournament size to 4. Furthermore, we have provided the functionality to change the tournament size during evolution runs. The thing is that decreasing this parameter reduces the pressure and increases the model's diversity, helping GA avoid getting trapped in a local optimum.

Crossover Operators: We employed three types of crossover in our project: Single-point crossover and Arithmetic crossover, which were covered during the lectures and practical classes, and Uniform crossover, which we implemented additionally. Uniform crossover involves randomly selecting the corresponding weight from either of the 2 parents for each gene (weight) into the offspring and ensures a good exploration of the search space. Though in some cases Uniform crossover may lead to disruptive changes in the weights, we decided to experiment with it, keeping in mind the simple nature of the NN architecture that we build.

Mutation Operators: In our project, we used two mutation operators. Firstly, we utilized the Inversion mutation, which was implemented during the practical classes. Additionally, since the weights of the neural network are continuous values, we developed an Arithmetic mutation. This mutation introduces a small perturbation to each weight by randomly adding a value within a chosen range. To ensure the perturbation remains within a reasonable scale, we selected a small range of random values between -0.1 and 0.1. This range aligns with the range of weights in the neural network, which falls between -1 and 1.

Additional Implementations: We made several enhancements to the code. First, in the Population class, we modified the 'evolve' method to not only print the representation and fitness value of the best individuals throughout generations but also return a list of these best fitness values once the evolution iterations are completed. Having the best fitness values for all generations helps us estimate the "typical" performance of a specific GA configuration on multiple runs.

In the main executable file, 'mnist.py', we implemented a separate function to perform independent runs for the desired algorithm configuration and store the results in a separate local .csv file.

To ensure the code's versatility for potential future use in other image classification problems, we tried to make our code general. In particular, we perform the loading of the MNIST images in the file called 'mnist_data.py'. Additionally, we defined NN parameters such as input/hidden/output layer sizes, activation functions, and accuracy metric at the beginning of the 'mnist.py' file instead of hardcoding them within the 'get_fitness' function. These changes allow for easier customization and adaptability of the code.

3. Benchmarking and Initial Analysis

A result of the 'No Free Lunch Theorem' implies there is no systematic way to select the most effective algorithm or even its optimal configuration. That is why through trials and comparative testing, we tried to find the optimal GA to solve the given maximization problem in two phases.

During the first phase, we explored and benchmarked multiple GA configurations in application to our problem. In particular, we tested 36 different configurations, summarised in the following table:

Selection	Tournament Size	Crossover	Crossover Probability	Mutation	Mutation Probability
fps	n.a.	Single Point Crossover	90%	Inversion Mutation	10% and 30%
fps	n.a.	Single Point Crossover	90%	Arithmetic Mutation	10% and 30%
fps	n.a.	Arithmetic Crossover	90%	Inversion Mutation	10% and 30%
fps	n.a.	Arithmetic Crossover	90%	Arithmetic Mutation	10% and 30%
fps	n.a.	Uniform Crossover	90%	Inversion Mutation	10% and 30%
fps	n.a.	Uniform Crossover	90%	Arithmetic Mutation	10% and 30%
Tournament Selection	2 and 4	Single Point Crossover	90%	Inversion Mutation	10% and 30%
Tournament Selection	2 and 4	Single Point Crossover	90%	Arithmetic Mutation	10% and 30%
Tournament Selection	2 and 4	Arithmetic Crossover	90%	Inversion Mutation	10% and 30%
Tournament Selection	2 and 4	Arithmetic Crossover	90%	Arithmetic Mutation	10% and 30%
Tournament Selection	2 and 4	Uniform Crossover	90%	Inversion Mutation	10% and 30%
Tournament Selection	2 and 4	Uniform Crossover	90%	Arithmetic Mutation	10% and 30%

Table 1 - 36 combinations tested during the first phase

To ensure a reasonable comparison, keeping in mind the computational limitations at hand, we executed 10 independent runs for each configuration. Each run encompassed a population of 20 individuals evolving over 20 generations, with the inclusion of elitism as an integral part of the implementation.

During the second phase, we further tested and analyzed the three most promising configurations identified during the initial phase. Given the non-deterministic nature of these models, which rely on random events, we ran the configurations that we deemed to be the most optimal for a total of 30 times. This choice aligns with statistical principles, as a sample size of 30 is widely recognized for its ability to yield more robust and conclusive results.

Upon executing all 36 selected configurations, we captured and stored the best fitness value for each generation in every run. Subsequently, using the code in the 'result_analysis.ipynb' notebook, we loaded and preprocessed the collected data, and calculated the Average Best Fitness (ABF) values along with the Standard Deviation values across generations. These metrics served as the basis for generating several plots, enabling us to gain valuable insights into the performance of the different configurations.

We present diagrams illustrating the progression of ABF across generations for each configuration in the Appendix to this report. Given the substantial total number of configurations, we adopted a multi-faceted approach to present the information from various perspectives, splitting by selection techniques, crossover methods, and mutation operators [A][C][E]. Additionally, we constructed boxplots that provided further insights into the standard deviation, maintaining the same viewpoints [B][D][F].

In addition, for the 20th generation results (the last ones for the initial phase of analysis), we display consolidated boxplots [G] and heatmaps [H] that bring even more insight into the performance of different operators.

The initial analysis of the line plots indicates that the developed algorithm effectively addresses the maximization problem at hand. This is evident from the consistent and progressive increase in the best fitness values across successive generations. Notably, a significant majority of the configurations achieve

an accuracy of over 90%. As for the specific GA operators and parameters, we have gleaned some potentially informative observations below.

In terms of the selection methods, while there is no method that stands out over others significantly, there is one specific configuration worth noting. The tournament selection method with a tournament size of 4 tends to demonstrate superior performance with a smaller deviations in most cases [A][B][G].

When it comes to crossover operators, based on the same results and visualization, it was concluded that the Uniform crossover (also known as n-point crossover), plays a key role in enhancing model performance [C][D][G][H].

Regarding the mutation operators, it is difficult to say whether any of the two is statistically better than the other [E][F]. In terms of the mutation probability, an analysis of [G] reveals an overlap in the boxplots, with a mutation probability of 30% outperforming the one of 10% on the 20th generation of the executed runs. To establish a more solid understanding, conducting additional trials would be advisable here. While a higher mutation probability carries the risk of gene destruction, a certain level of it is necessary to ensure the algorithm evolves towards maximizing the objective.

4. Final Solution

Based on the preceding analysis, we opted to select the three most promising configurations for additional trials, with the purpose of either uncovering more statistical substantiation or verifying the correctness of our initial conclusions with more runs.

Selection	Tournament Size	Crossover	Crossover Probability	Mutation	Mutation Probability
Tournament Selection	4	Single Point Crossover	90%	Inversion Mutation	30%
Tournament Selection	4	Arithmetic Crossover	90%	Arithmetic Mutation	30%
Tournament Selection	4	Uniform Crossover	90%	Arithmetic Mutation	30%

Table 2 - Three Final configurations.

We conducted a total of 30 independent runs for each of the three configurations. In each run, we employed a population size of 20 individuals evolving over 20 generations. The Average Best Fitness (ABF) values for the final configurations were calculated as previously and are presented in the plot [I].

From the analysis of the plot, it is evident that among the three algorithms under consideration, the one utilizing Uniform Crossover with Arithmetic Mutation delivered the best results. The ABF values for this configuration exhibit a rapid ascent, reaching proximity to 1 by the 10th generation on average.

For a more detailed examination of the ABF values across generations, please refer to the corresponding boxplot [J]. The boxplot further confirms the swift convergence of this particular combination towards the global maximum.

5. Conclusion and Discussion

To find the optimal GA configuration for the MNIST image classification problem (a maximization problem in terms of categorical accuracy), we conducted a two-phase experimentation process. In the first phase, we explored 36 different configurations, ensuring a fair comparison of implemented operators by conducting 10 independent trials for each configuration. In the second phase, we selected the three most promising configurations based on the results of the initial phase and performed 30 independent runs for each configuration to obtain more robust and scientifically significant data.

Our analysis revealed that the configuration utilizing Uniform crossover with Arithmetic mutation yielded the best performance results. In particular, the ABF values obtained for this configuration showed significant improvement during the population evolution, reaching an average proximity close to 1 by the 10th generation.

While the configuration obtained during the project implementation appeared to produce exceptionally high fitness values, the frequent achievement of 100% categorical accuracy on the training dataset raises valid concerns regarding overfitting and the model's ability to generalize to unseen data. To address this concern, we decided to conduct at least one trial run for the best-performing GA configuration, while also calculating the accuracy on the test dataset at each generation. The output results can be found in Appendix [K].

Remarkably, the model exhibited accurate predictions on unseen data, indicating successful optimization of the neural network weights using GA algorithms. The high performance on unseen data can be partly attributed to the simplicity of the MNIST dataset and the potential compatibility of the Uniform crossover operator with the MNIST classification problem. The exploration capabilities of Uniform crossover enable the discovery of diverse solutions within the search space, which proves advantageous in capturing the straightforward and distinct patterns present in MNIST.

6. Division of Labor

Both students contributed equally to the project.

7. References

[1] Vannesch. L., Silva S., *Lectures on Intelligent Systems*, 2023

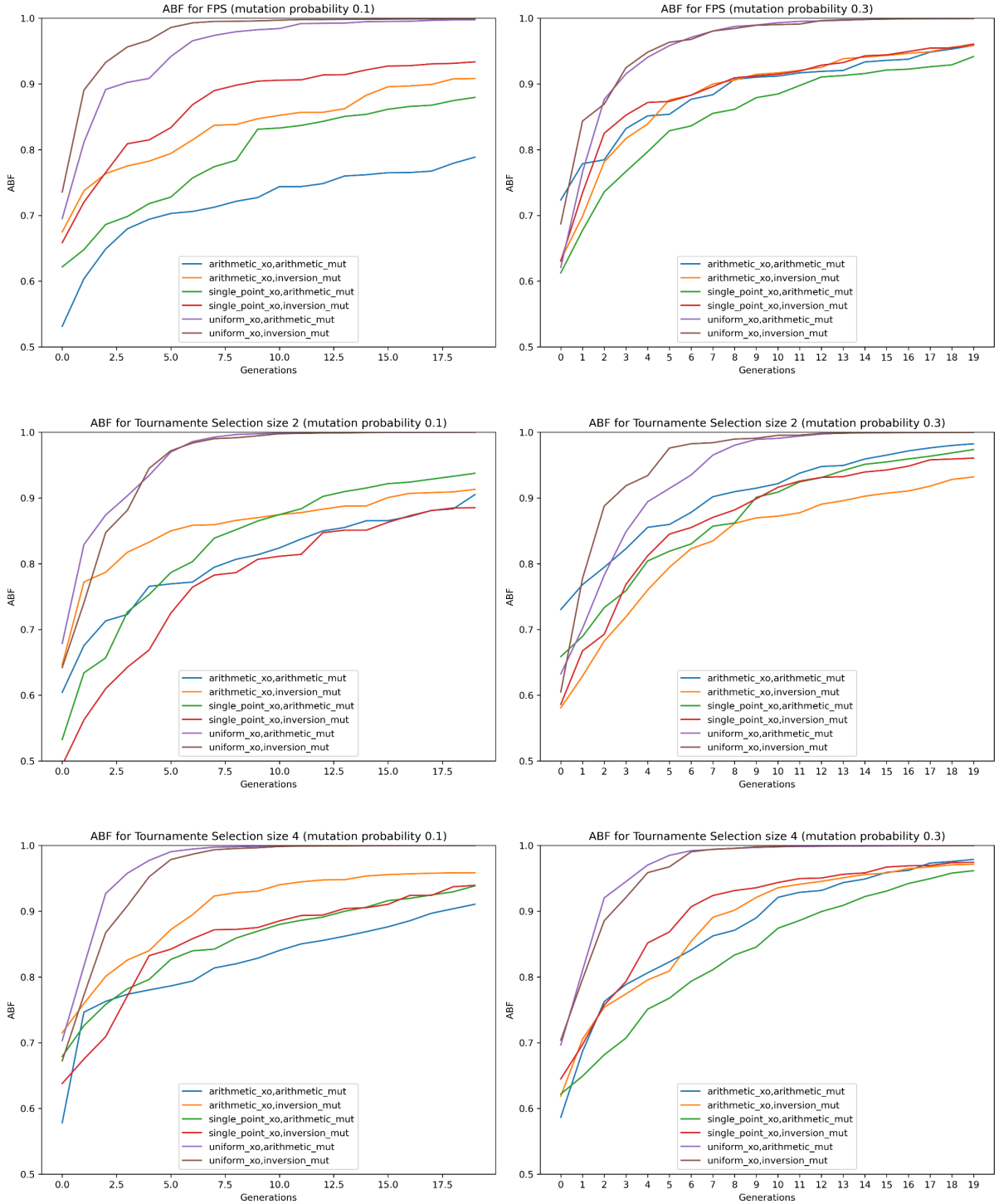
[2] Library “Charles” developed during practical classes

[3] Grag A. (2021), Crossover Operator in Genetic Algorithm, 2021, https://medium.com/geekculture/crossover-operators-in-ga-cffa77cdd0c8, retrieved May 25 2023

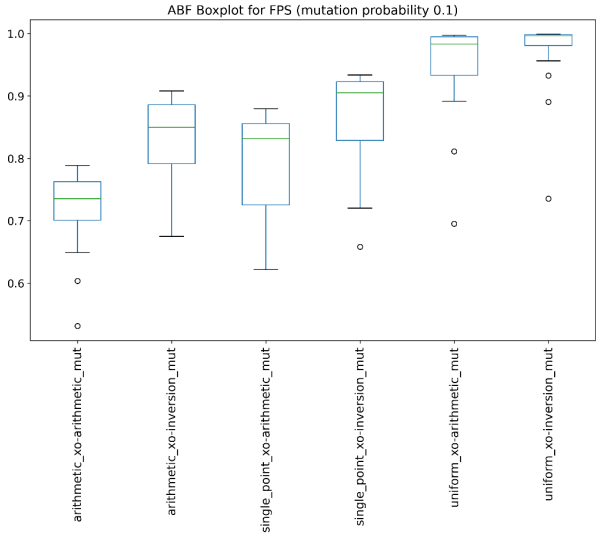
[4] Genetic Algorithms - Crossover, https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm, retrieved May 25 2023

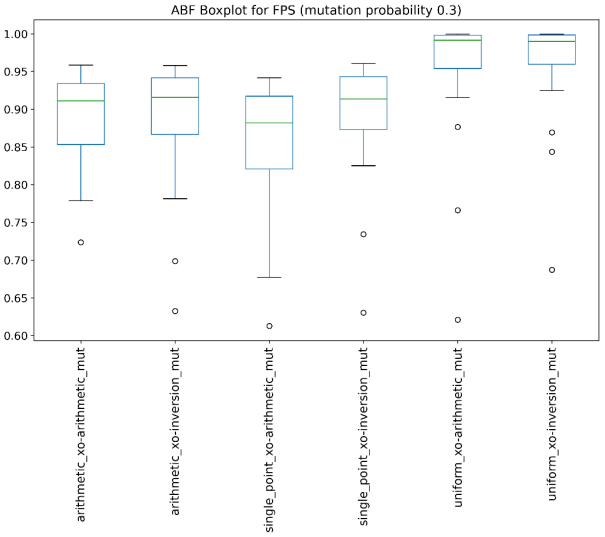
8. Appendix

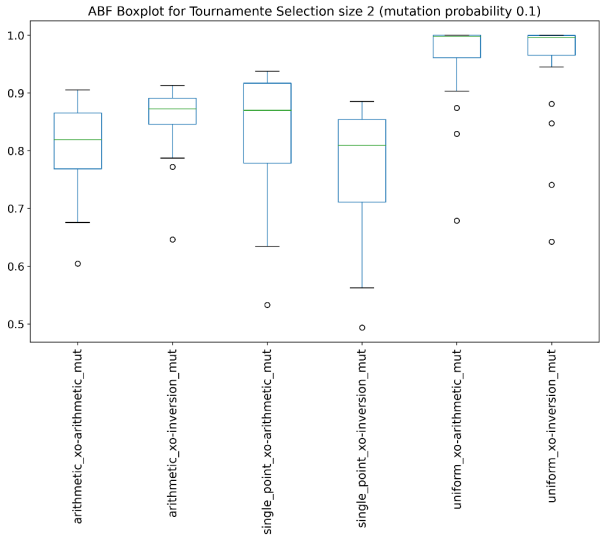
[A] - Selection - LinePlot

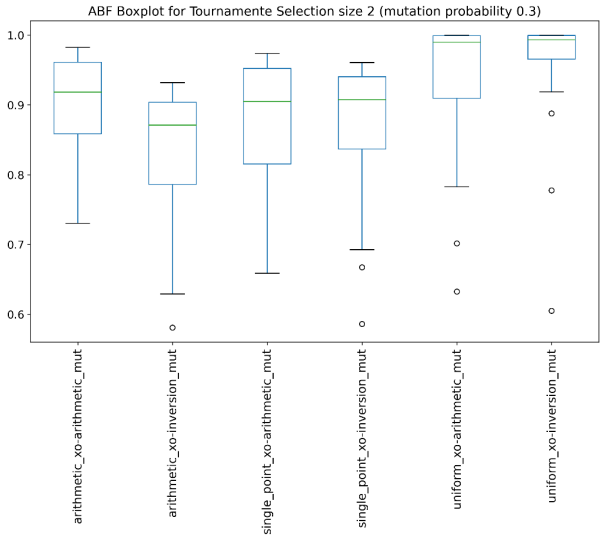


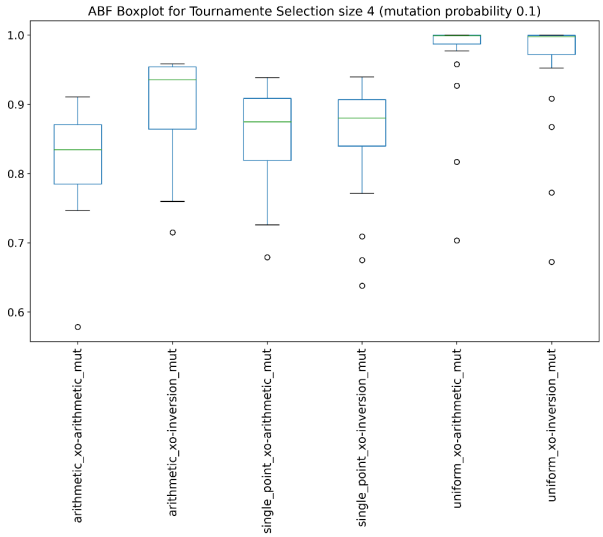
[B] - Selection - BoxPlot

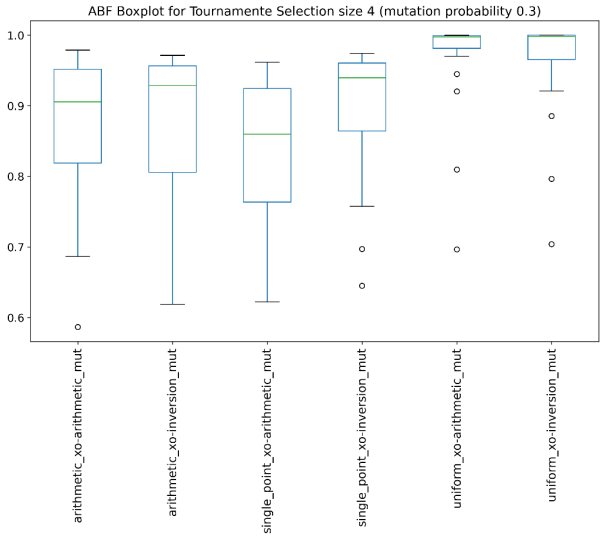




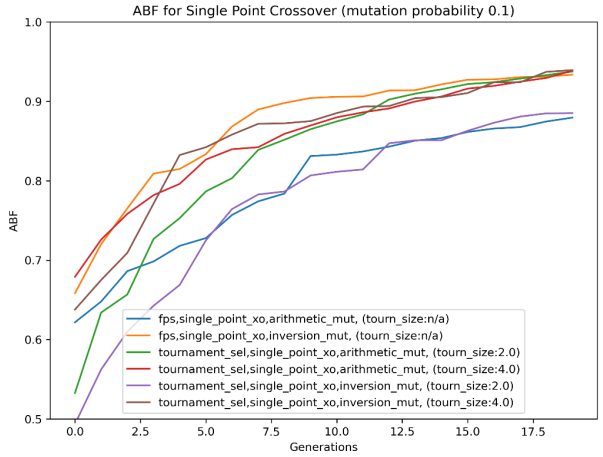


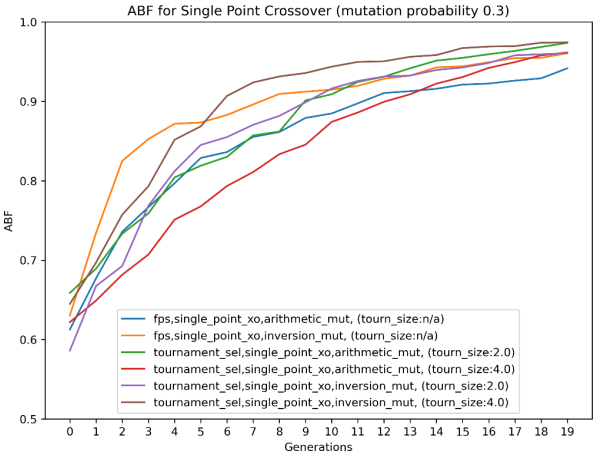


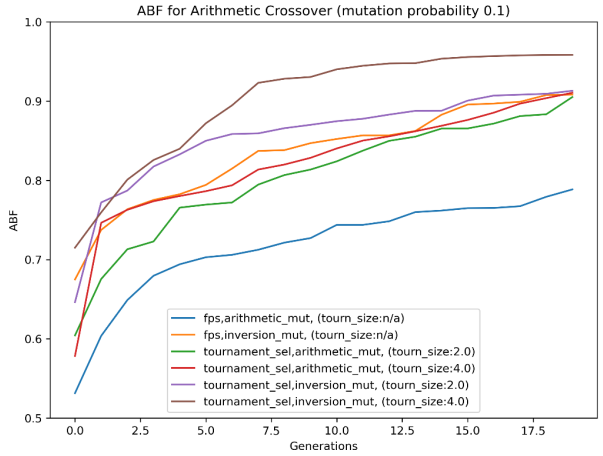


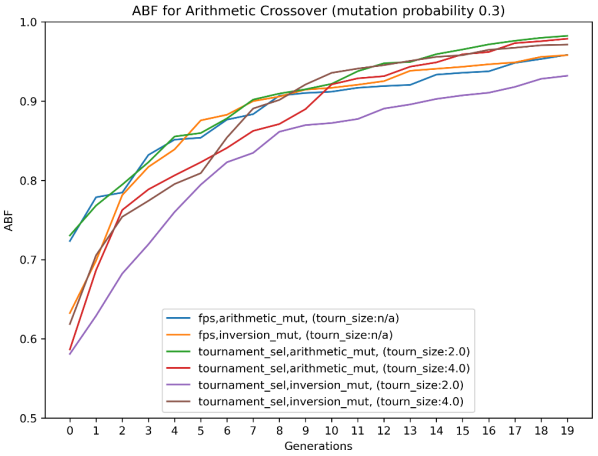


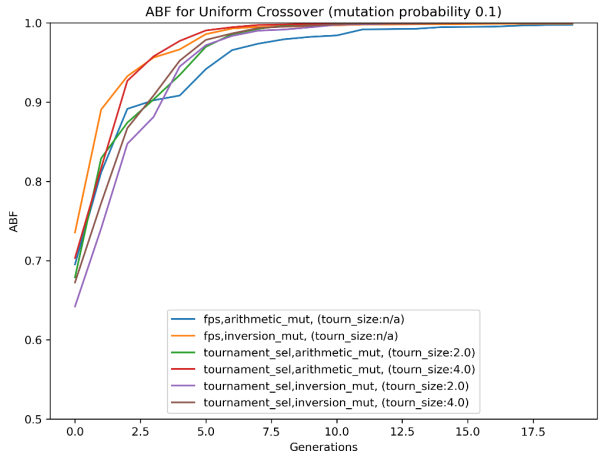
[C] - Crossover - LinePlot

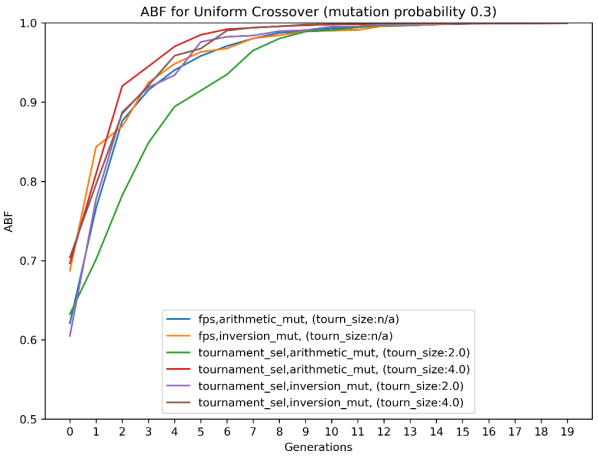




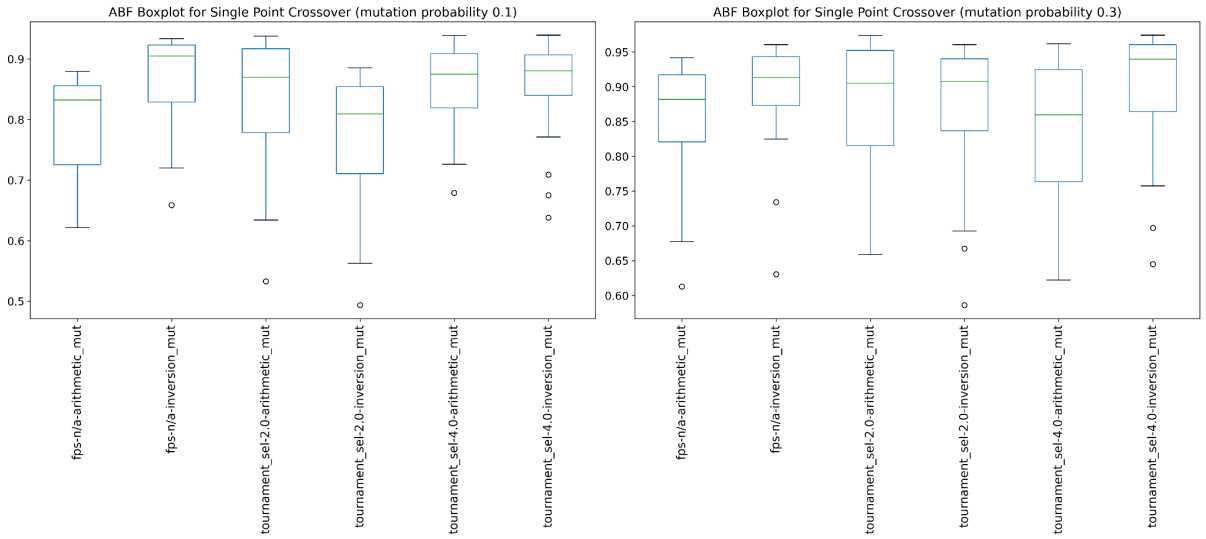


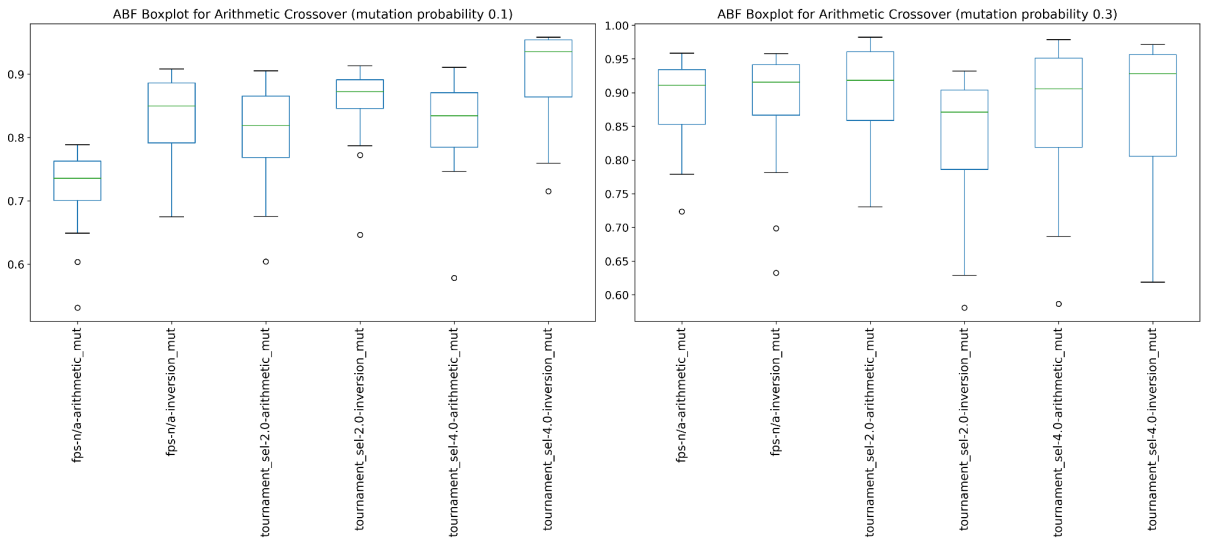


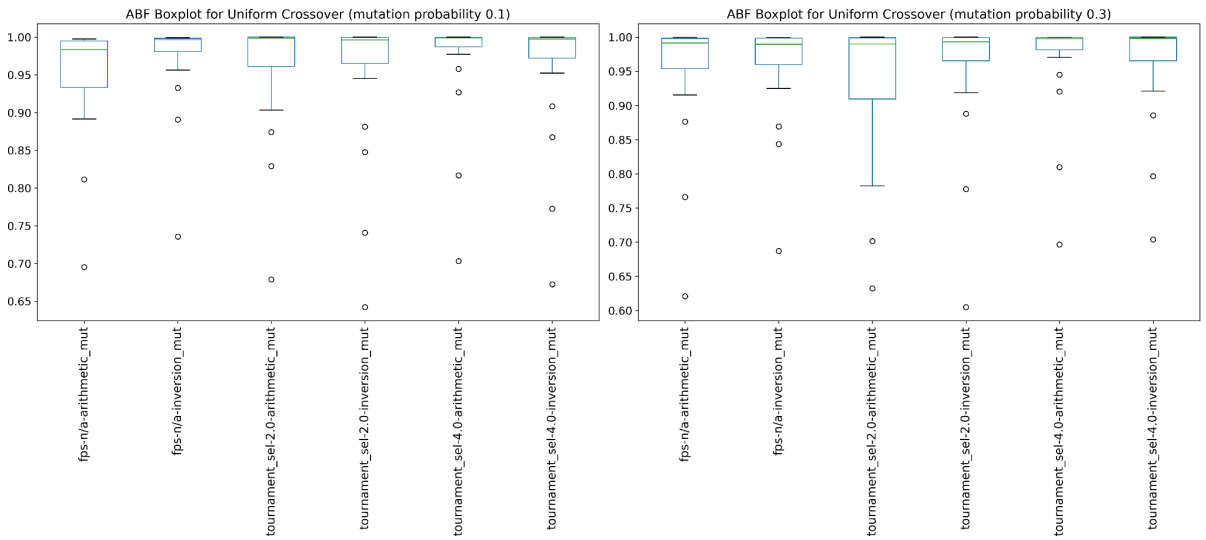




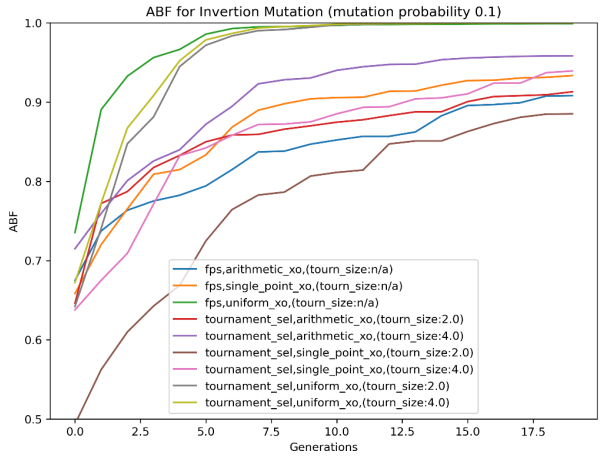
[D] - Crossover - BoxPlot

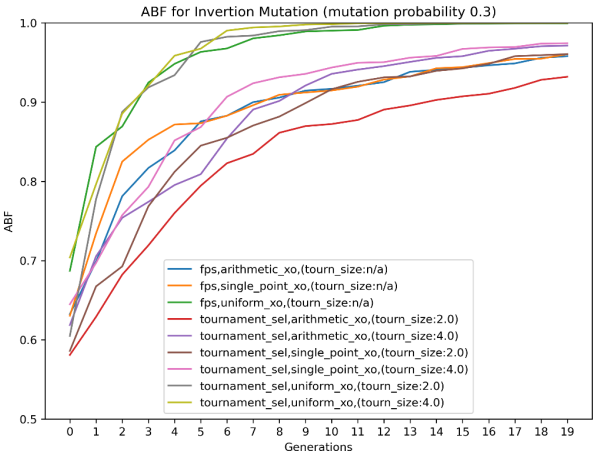


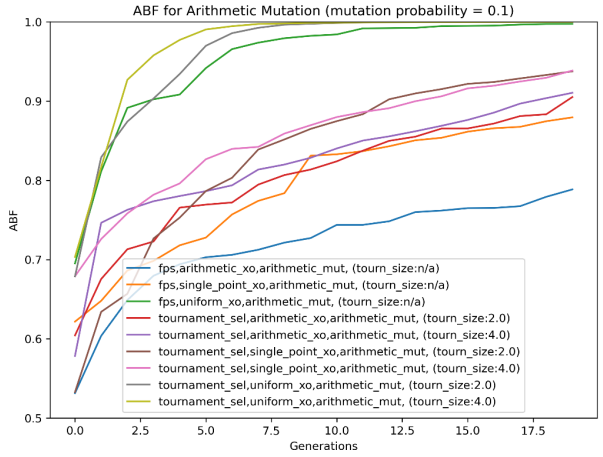


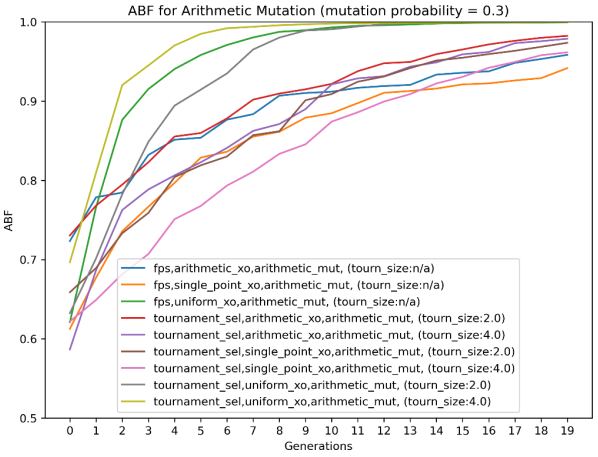


[E] - Mutation – LinePlot

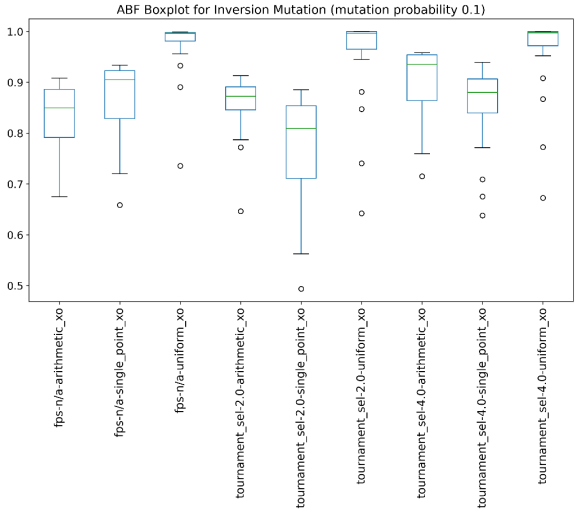


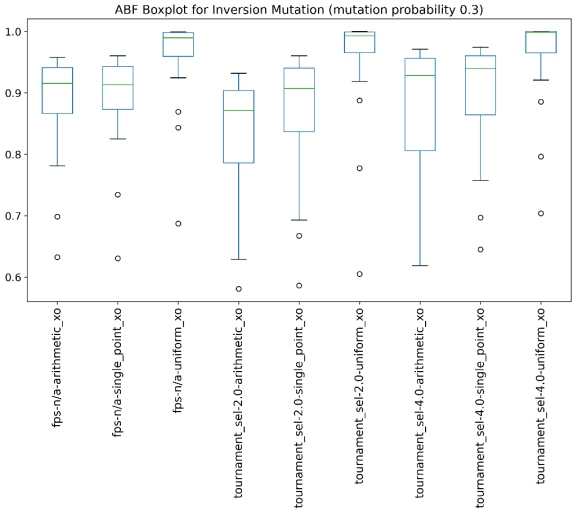


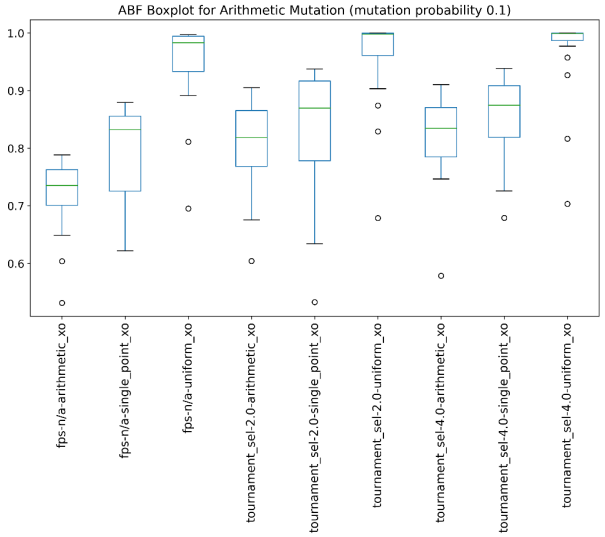


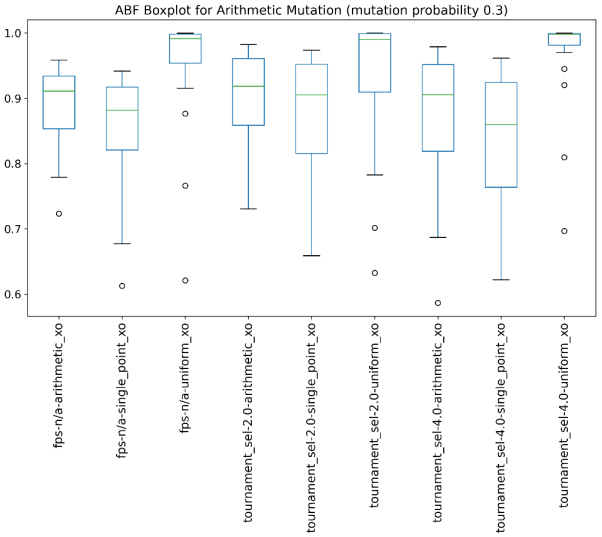


[F] - Mutation - BoxPlot

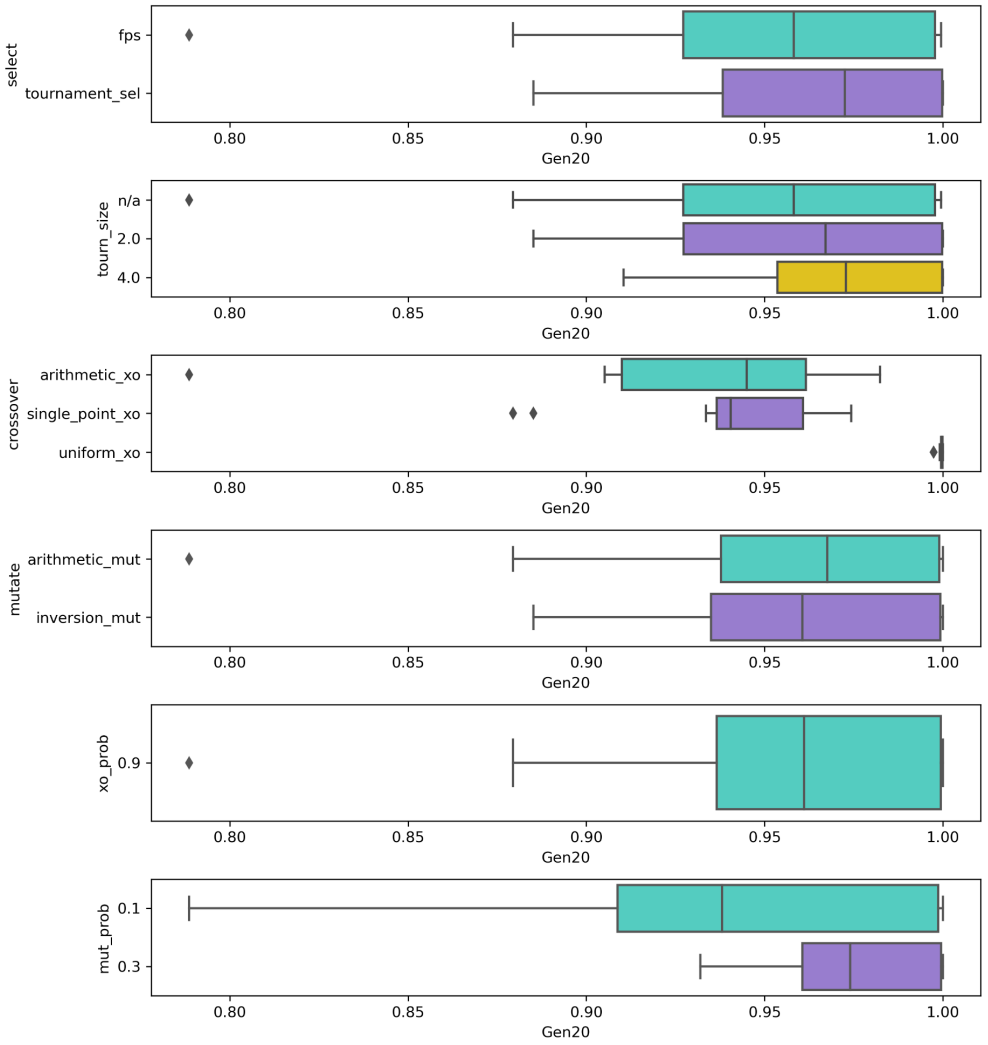






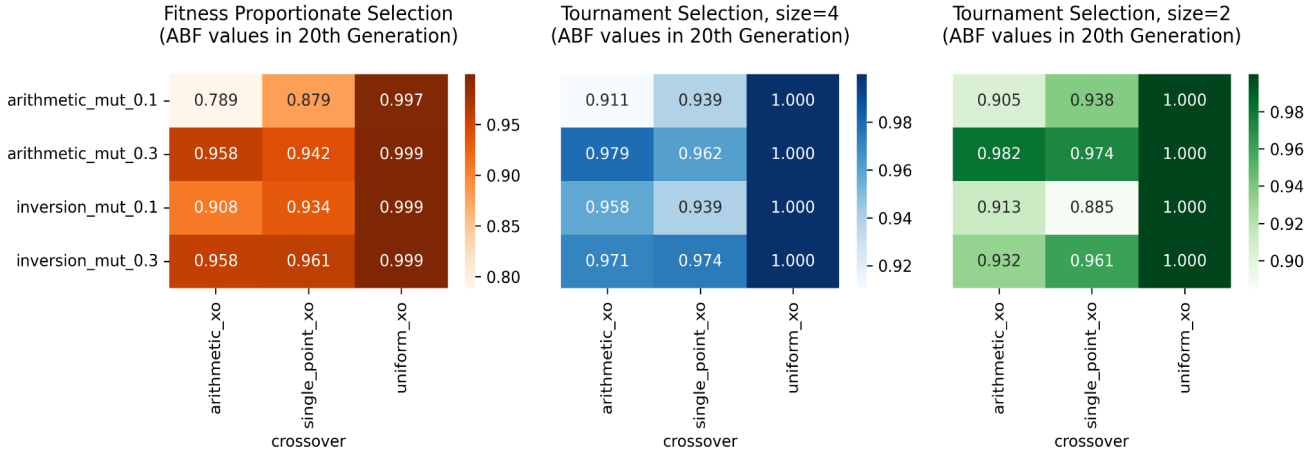


[G] - Comparison results

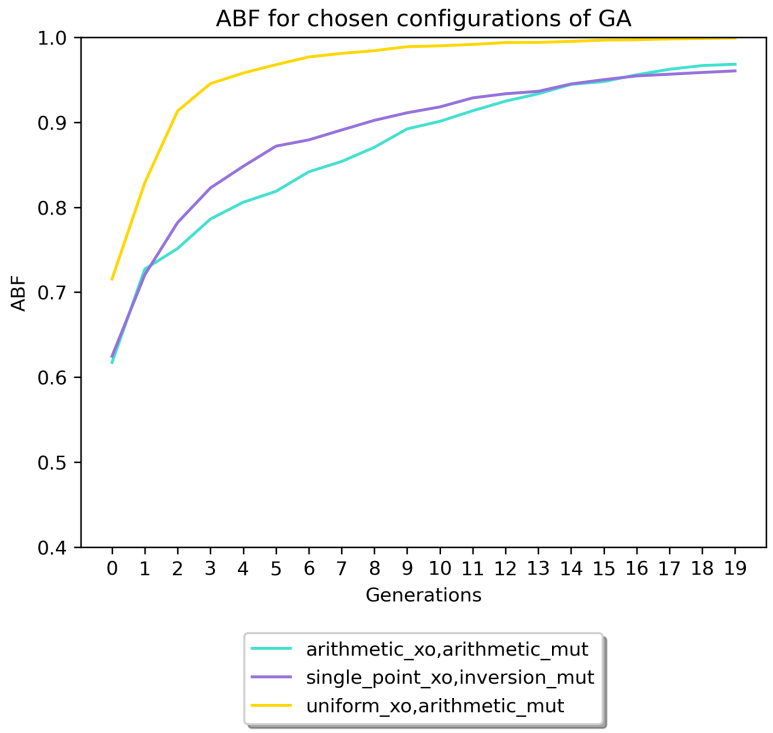


ABF value distibution in 20th Generation

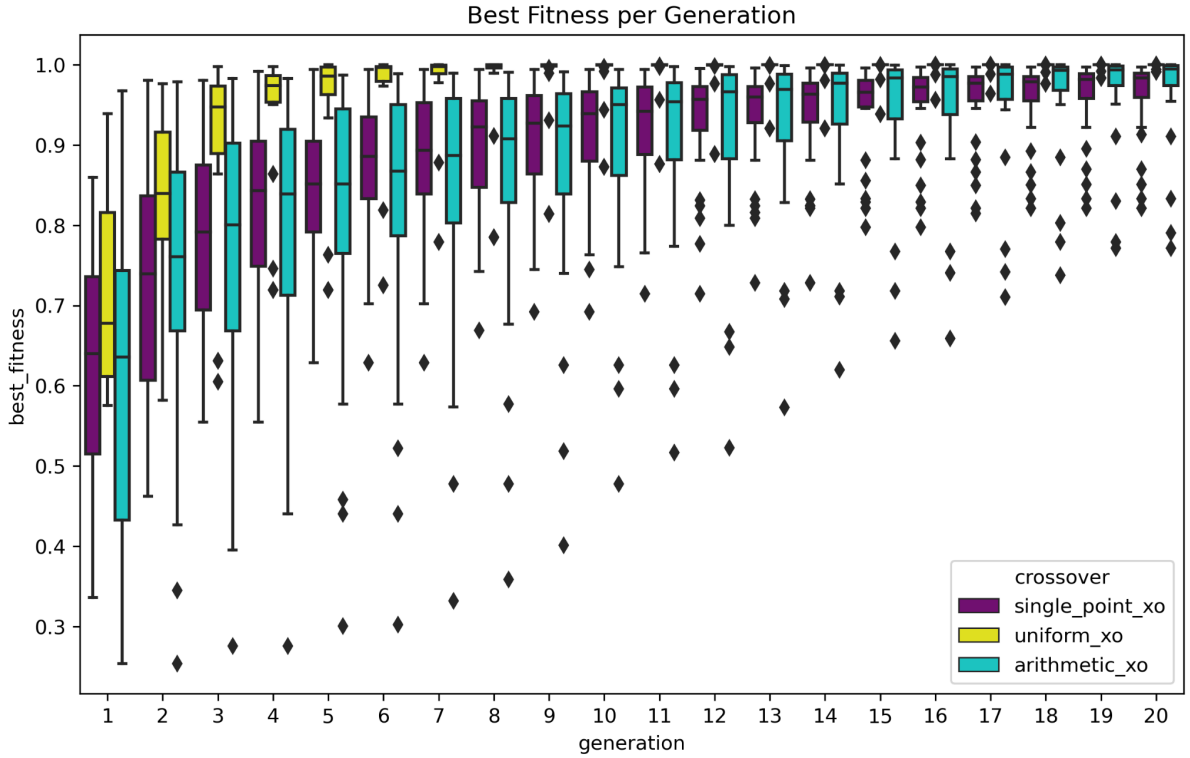
[H] Heatmaps



[I] ABF for final configurations observed for 30 independent runs



[J] ABF Boxblots split by Generation



[K] Output Log to Check Overfitting

```
Generating initial population...
100%|██████████| 20/20 [01:40<00:00,  5.05s/it]
Evolving...
  0%|          | 0/20 [00:00<?, ?it/s]Starting Generation  1 ...
  5%|▌         | 1/20 [01:51<35:19, 111.53s/it]Best Individual in Generation 1:
Individual: 784, 512, 10; Fitness: 0.7801166772842407; Test_fitness: 0.772599995136261;
 10%|█         | 2/20 [03:46<34:03, 113.52s/it]Best Individual in Generation 2:
Individual: 784, 512, 10; Fitness: 0.7801166772842407; Test_fitness: 0.772599995136261;
Starting Generation  3 ...
 15%|█▌        | 3/20 [05:30<30:58, 109.30s/it]Best Individual in Generation 3:
Individual: 784, 512, 10; Fitness: 0.7801166772842407; Test_fitness: 0.772599995136261;
Starting Generation  4 ...
 20%|██        | 4/20 [07:18<28:58, 108.64s/it]Best Individual in Generation 4:
Individual: 784, 512, 10; Fitness: 0.9191833138465881; Test_fitness: 0.9194999933242798;
Starting Generation  5 ...
 25%|██▌       | 5/20 [08:56<26:14, 104.94s/it]Best Individual in Generation 5:
Individual: 784, 512, 10; Fitness: 0.9749166369438171; Test_fitness: 0.9775999784469604;
Starting Generation  6 ...
 30%|███       | 6/20 [10:43<24:39, 105.66s/it]Best Individual in Generation 6:
Individual: 784, 512, 10; Fitness: 0.9749166369438171; Test_fitness: 0.9775999784469604;
Starting Generation  7 ...
 35%|███▌      | 7/20 [12:23<22:28, 103.76s/it]Best Individual in Generation 7:
Individual: 784, 512, 10; Fitness: 0.9909999966621399; Test_fitness: 0.9926999807357788;
Starting Generation  8 ...
 40%|████      | 8/20 [14:09<20:54, 104.51s/it]Best Individual in Generation 8:
Individual: 784, 512, 10; Fitness: 0.9909999966621399; Test_fitness: 0.9926999807357788;
Starting Generation  9 ...
 45%|████▌     | 9/20 [15:57<19:21, 105.63s/it]Best Individual in Generation 9:
Individual: 784, 512, 10; Fitness: 0.9986666440963745; Test_fitness: 0.9991000294685364;
Starting Generation  10 ...
 50%|█████     | 10/20 [17:36<17:15, 103.57s/it]Best Individual in Generation 10:
Individual: 784, 512, 10; Fitness: 0.9987000226974487; Test_fitness: 0.9987999796867371;
Starting Generation  11 ...
 55%|█████▌    | 11/20 [19:19<15:30, 103.37s/it]Best Individual in Generation 11:
Individual: 784, 512, 10; Fitness: 0.9997333288192749; Test_fitness: 0.9995999932289124;
Starting Generation  12 ...
 60%|██████    | 12/20 [21:02<13:45, 103.20s/it]Best Individual in Generation 12:
Individual: 784, 512, 10; Fitness: 0.9997333288192749; Test_fitness: 0.9995999932289124;
```

```
Starting Generation  13 ...
 65%|██████▌   | 13/20 [22:42<11:54, 102.14s/it]Best Individual in Generation 13:
Individual: 784, 512, 10; Fitness: 0.9998499751091003; Test_fitness: 1.0;
Starting Generation  14 ...
 70%|███████   | 14/20 [24:25<10:15, 102.61s/it]Best Individual in Generation 14:
Individual: 784, 512, 10; Fitness: 0.9999833106994629; Test_fitness: 1.0;
Starting Generation  15 ...
 75%|███████▌  | 15/20 [26:12<08:39, 103.84s/it]Best Individual in Generation 15:
Individual: 784, 512, 10; Fitness: 0.9999833106994629; Test_fitness: 1.0;
Starting Generation  16 ...
 80%|████████  | 16/20 [27:52<06:50, 102.57s/it]Best Individual in Generation 16:
Individual: 784, 512, 10; Fitness: 1.0; Test_fitness: 1.0;
Starting Generation  17 ...
 85%|████████▌ | 17/20 [29:29<05:03, 101.03s/it]Best Individual in Generation 17:
Individual: 784, 512, 10; Fitness: 1.0; Test_fitness: 1.0;
Starting Generation  18 ...
 90%|█████████ | 18/20 [31:08<03:20, 100.48s/it]Best Individual in Generation 18:
Individual: 784, 512, 10; Fitness: 1.0; Test_fitness: 0.9998999834060669;
Starting Generation  19 ...
 95%|█████████▌| 19/20 [32:44<01:39, 99.06s/it] Best Individual in Generation 19:
Individual: 784, 512, 10; Fitness: 1.0; Test_fitness: 0.9998999834060669;
Starting Generation  20 ...
100%|██████████| 20/20 [34:27<00:00, 103.39s/it]Best Individual in Generation 20:
Individual: 784, 512, 10; Fitness: 1.0; Test_fitness: 1.0;
```