Evolutionary computation
Solution to the eight queen's problem with evolutionary algorithm

General:
The eight queen's problem is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other; thus, a solution requires that no two queens share the same row, column, or diagonal.

Goal:
Solve the eight queen's problem with genetic algorithm.

Tools:
1) Programming language – Python.
2) Evolutionary computation framework – DEAP.

First experiment (Array)

Genetic algorithm parameters:

Representation	array (size = 8)
Recombination	single point crossover
Recombination probability	70%
Mutation	one-point mutation
Mutation probability	0.1%
Parent selection	fitness-proportional, roulette-wheel sampling
Population size	100 individuals
Initialization	random
Termination condition	100 generations

Process of work:

1. From Phenotype to Genotype

To start with we need to define the genotype.

- In our task, the *phenotype* will represent the chessboard configuration with the queen's positions.
- The *genotype* will be encoded in the form of an array where the index and value will describe the position of the queen on the board (line and row). The values in the array may be repeated.

2. Initialization

Initialization of the initial *population* - at the first step, a set of candidates in the amount of 100 is generated for the solution, which gives new offspring as a result of the selection, mutation, and crossover operators.

3. Fitness Evaluation

The next step is the evaluation of candidates. To evaluate the fitness parameter, we will create a function (*evaluation()*) that will estimate the number of queens on the board that can be attacked. The lower this measure, the better a phenotype and zero is the best solution (optimal value).
To select chromosomes for crossing, we use *roulette-wheel sampling* (selection operator) (see the next step), the result of which is directly proportional to fitness score. Therefore, individuals with a high fitness score have a higher chance of being selected.
Since our task is to increase the fitness score (maximization), we will invert the score(the number of queens attacked):

$$F(P) = \frac{1}{P+1},$$

where: *F(P)* - Fitness Evaluation, *P* - number of queens attacked, *P=0* - optimal value.

4. Parent selection

The selection operator selects representatives for the use of genetic operators. For selection, we use *roulette-wheel sampling*. In this method, the probability of being selected directly depends on the Fitness function. Highly rated fitness individuals are more likely to be selected for crossover.

5. Crossover

We use *one-point crossover*, in this crossover, a random crossover point is selected and the tails of its two parents are swapped to get new offsprings. Resulting in two new representatives, with the genes of the first and second parent.

6. Mutation

The next step is the mutation operator (*mut_one_point*()). One gene of a random representative is mutated.

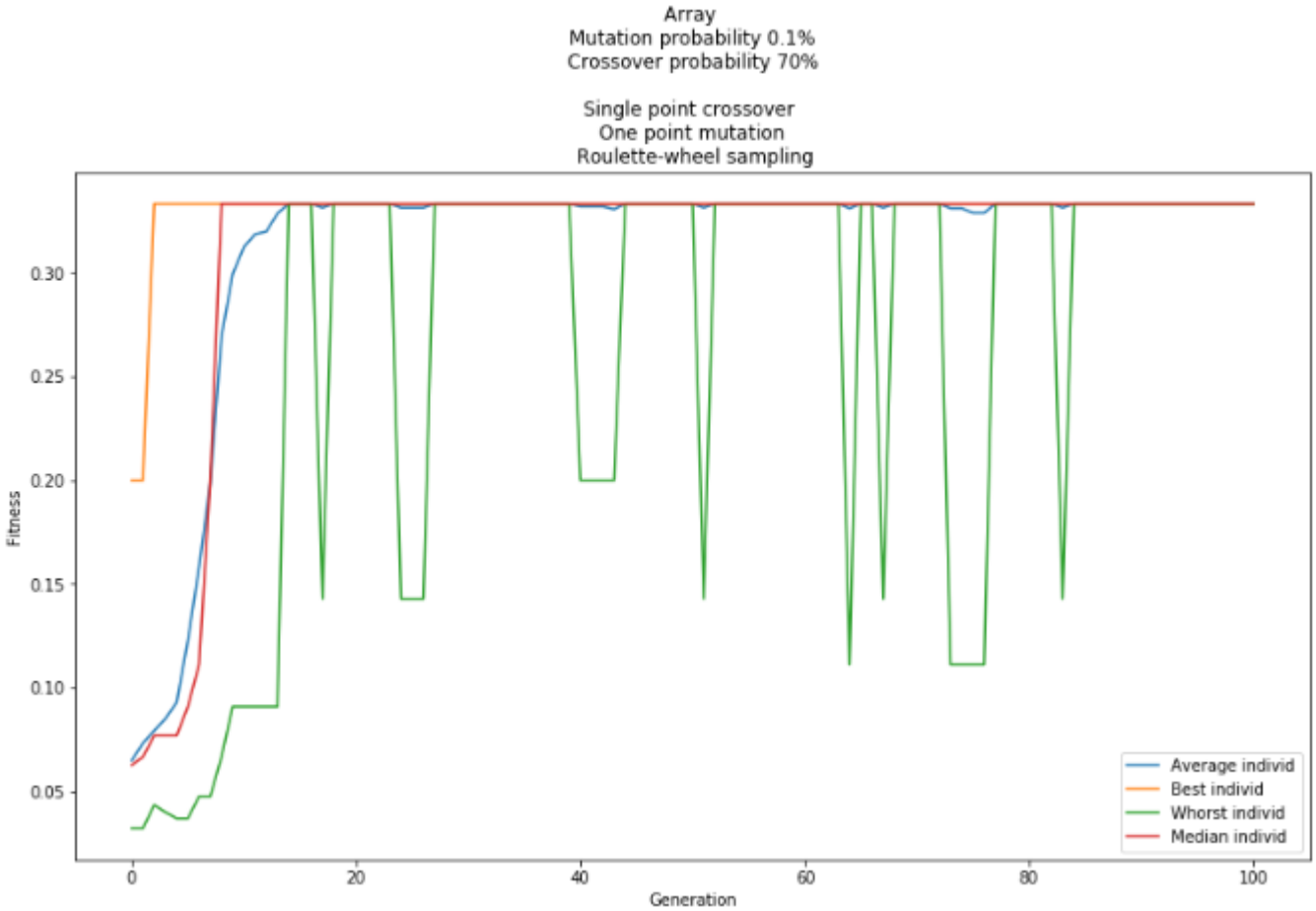
7. Final

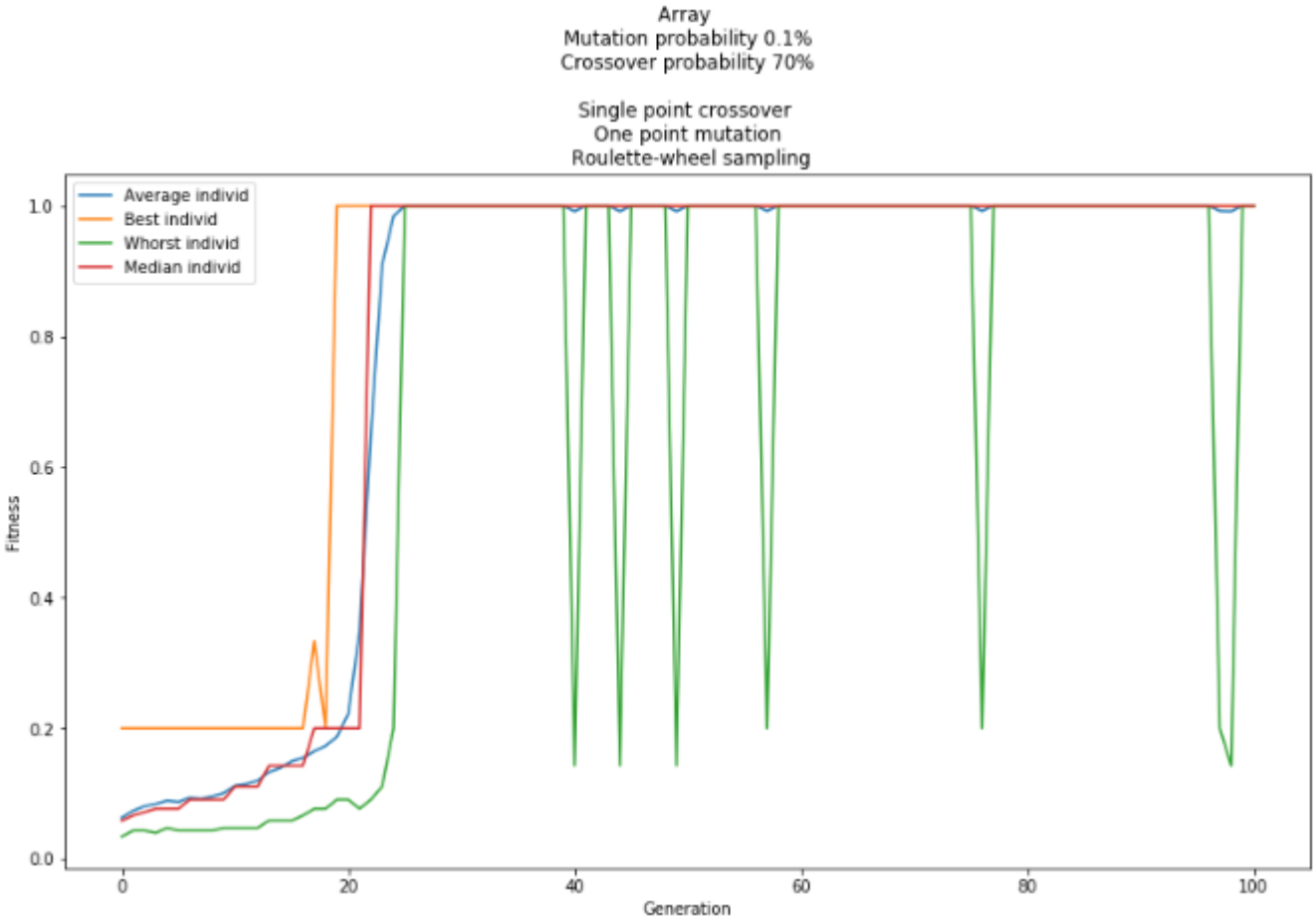
The final stage of each iteration is the formation of a new population. Stop occurs on the hundredth generation.

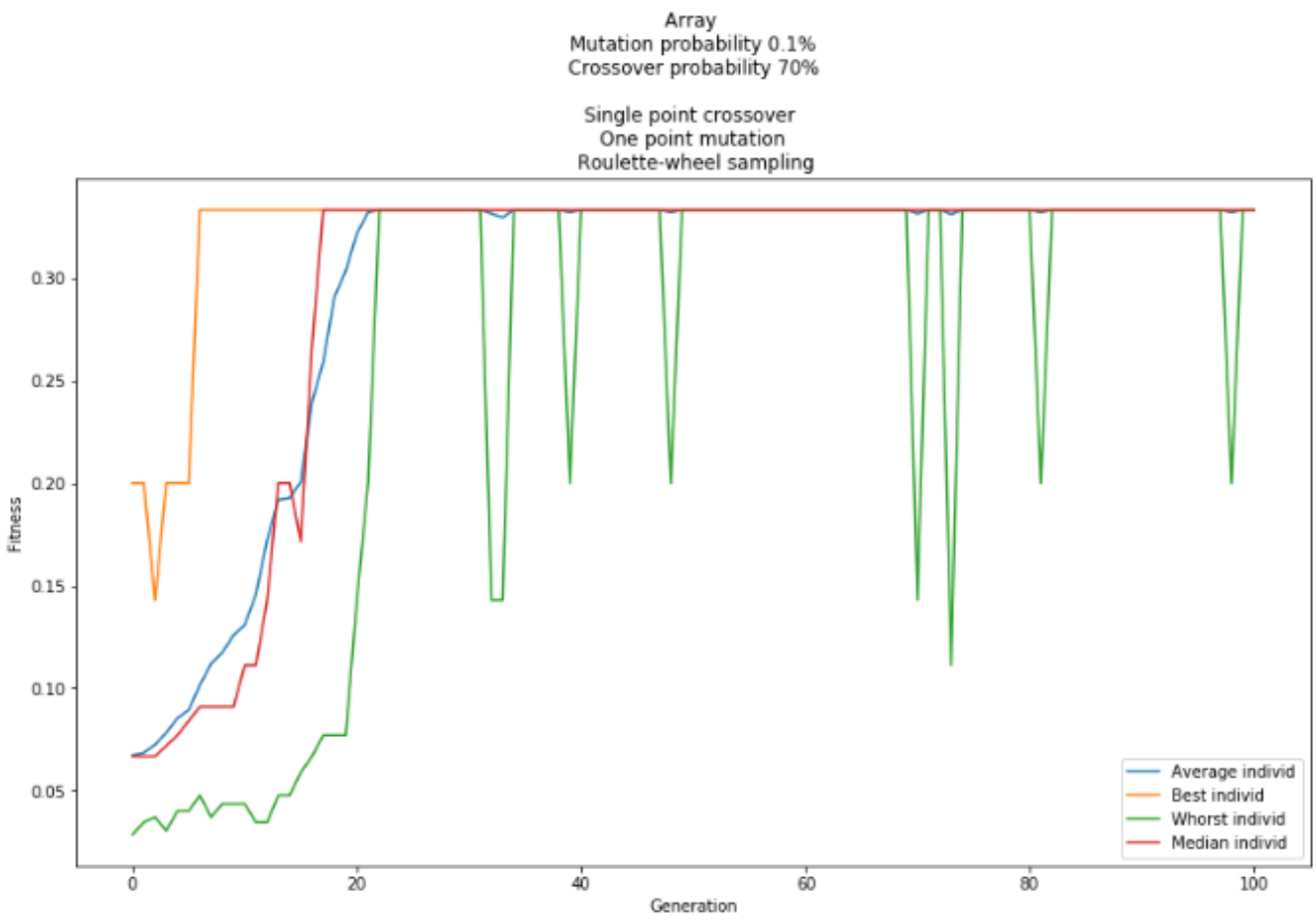
Results:

During this experiment, we made three runs:

Average time for termination: 1.09s







- Only one of the three runs managed to find the right solutions - which can be seen on the graph as convergence to fitness value of 1. In other cases, in the end we converge to a local maximum which is less than 1.

- As we see after the 20th generation, the entire population becomes approximately the same (the same fitness value). Individuals come to one certain value of fitness function, which does not change. In the future, a mutation of only one gene with a 0.1% probability of a mutation cannot lead to strong changes in the population as a whole, only in point individuals does fitness deteriorate. The crossover in this case does not affect, since the entire population is the same.

Second experiment (Permutation)

Genetic algorithm parameters:

Representation	permutation				
Recombination	partially mapped crossover (PMX)				
Recombination probability	70%	40%	10%	70%	70%
Mutation	swap				
Mutation probability	0.1%	0.1%	0.1%	1%	10%
Parent selection	fitness-proportional, roulette-wheel sampling				
Population size	100 individuals				
Initialization	random				
Termination condition	100 generations				

Process of work:

1. Initialization

Firstly we initialize a random population of chromosome of length 100. Every chromosome here is actually a randomly generated permutations of length 8.

2. Fitness Evaluation

The next step is the evaluation of candidates. To evaluate the fitness parameter, we will create a function (evaluation()) that will estimate the number of queens on the board that can be attacked. The lower this measure, the better a phenotype and zero is the best solution (optimal value). Since our task is to increase the fitness score (maximization), we will invert the score:

$$F(P) = \frac{1}{P+1},$$

where: *F(P)* - Fitness Evaluation, *P* - number of queens attacked, *P=0* - optimal value.

3. Parent selection

The selection operator selects representatives for the use of genetic operators. For selection, we use *Tournament selection*. Selected candidates are then passed on to the next generation. In a 2-way tournament selection, we select 2-individuals and run a tournament among them. Only the fittest candidate amongst those selected candidates is chosen and is passed on to the next generation. In this way many such tournaments take place and we have our final selection of candidates who move on to the next generation.

4. Crossover

We use a *partially matched crossover (PMX*) on the input individuals. The two individuals are modified in place.

5. Mutation

The next step is the mutation operator *(swap_mut*()). Two random genes are reversed.

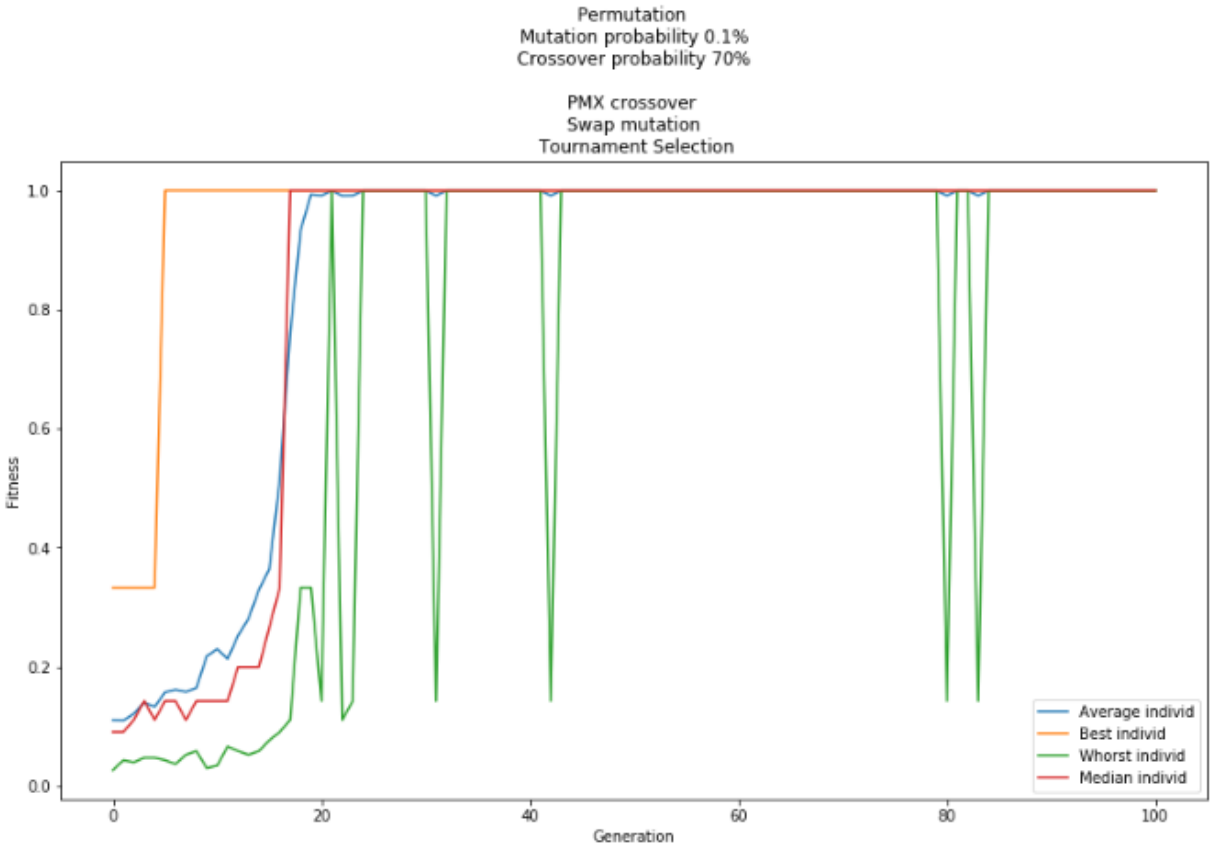
6. Final

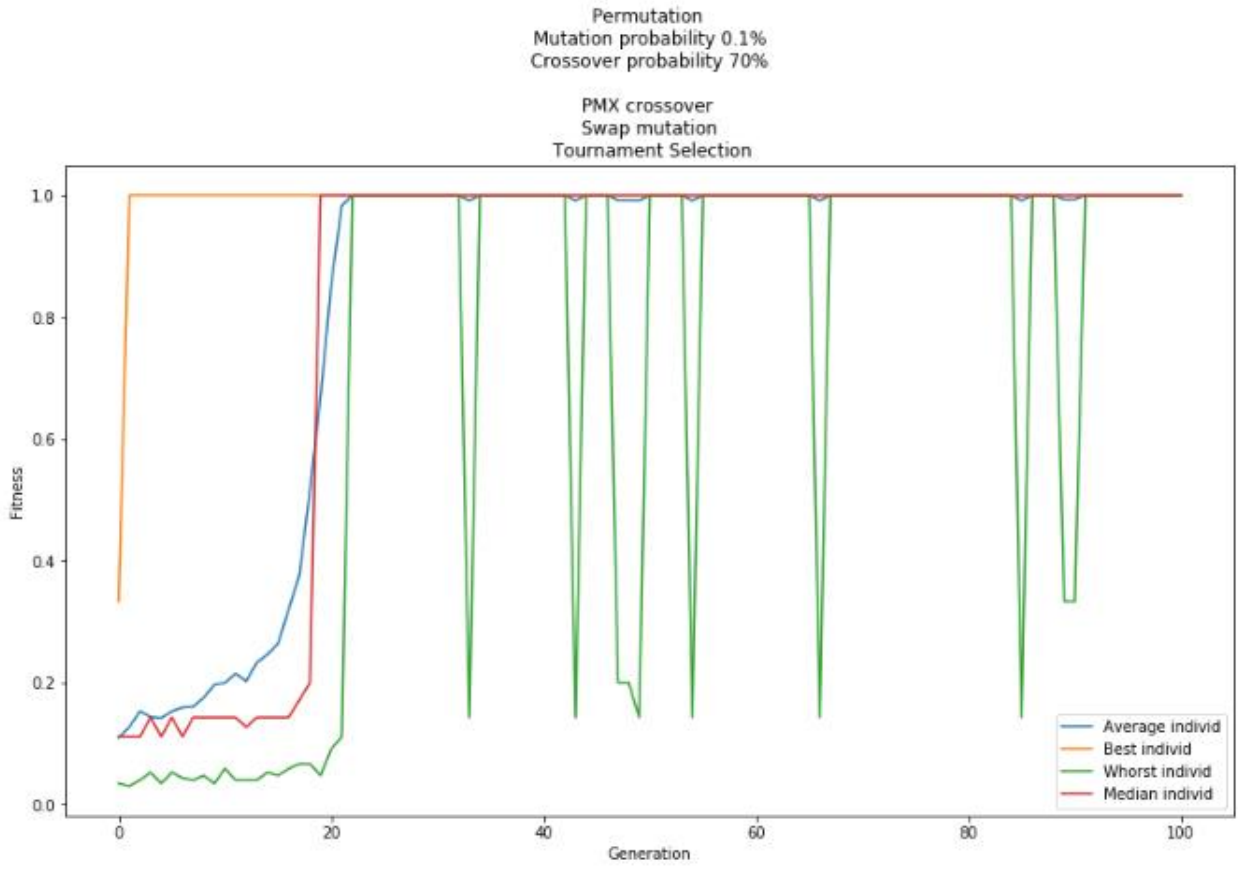
The final stage of each iteration is the formation of a new population. Stop occurs on the hundredth generation.

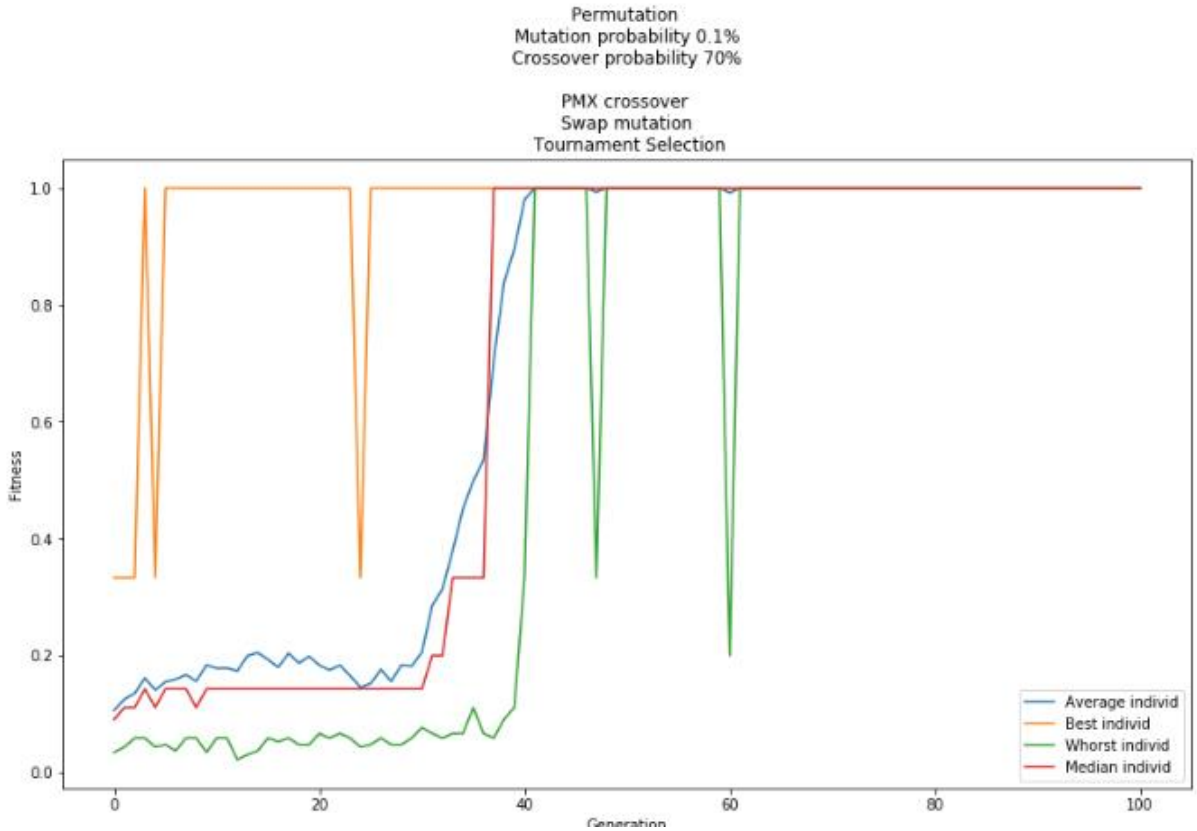
Results:

During this experiment, we made three runs for each of their three sets of mutation and crossover parameters:

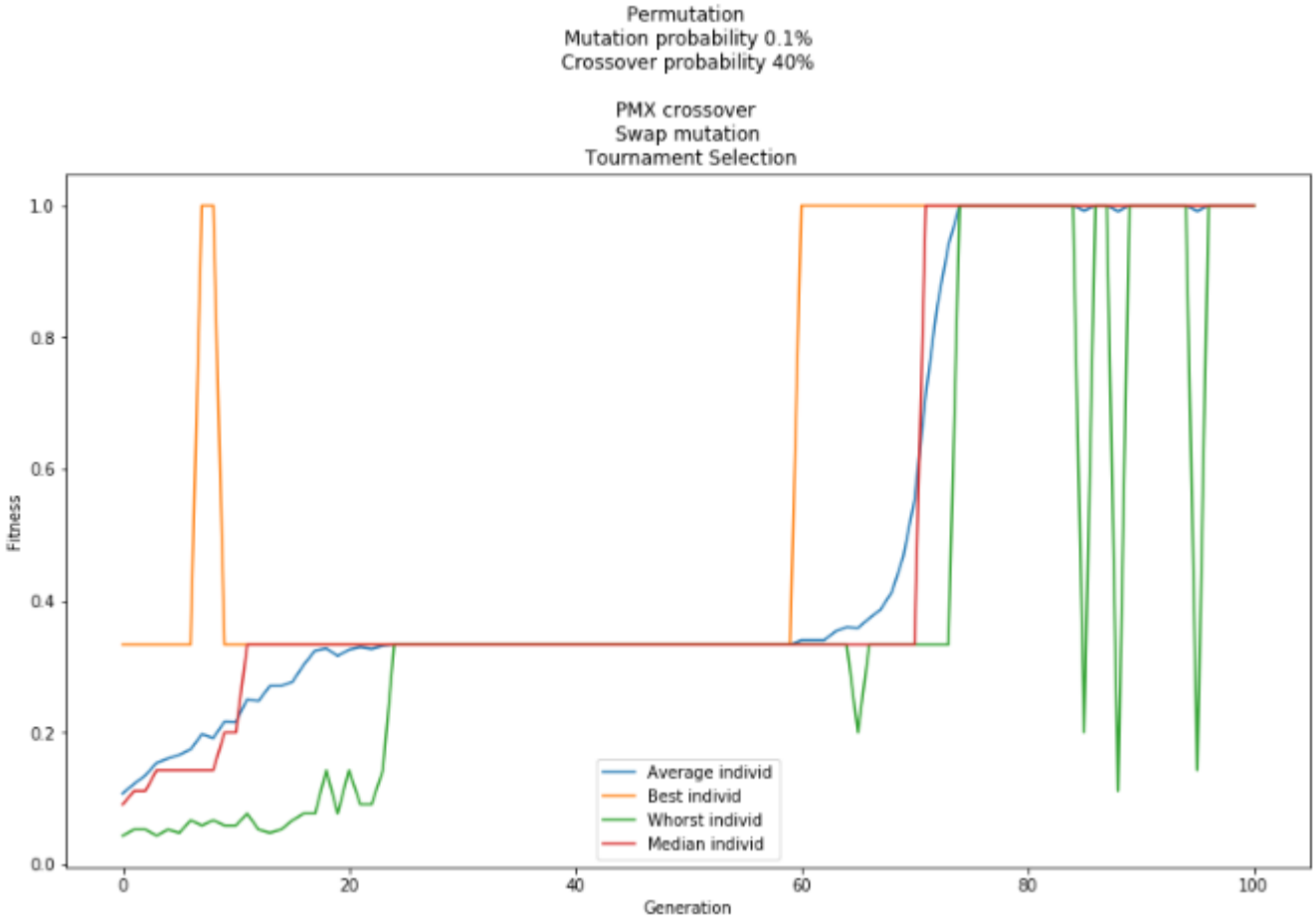
I. Running configuration with probabilities p_c=0.7, p_m=0.001:
Average time for termination: 0.47s





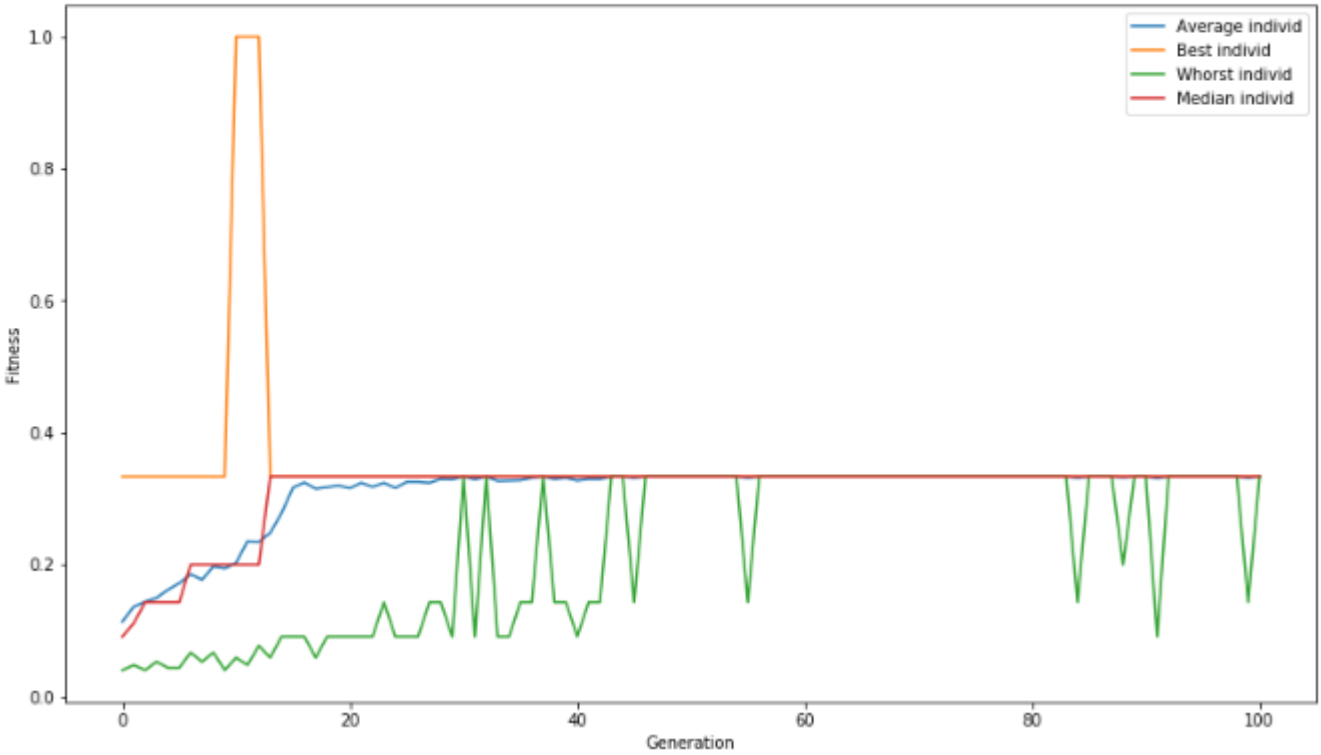


II. Running configuration with probabilities p_c=0.4, p_m=0.001:
Average time for termination: 0.71s



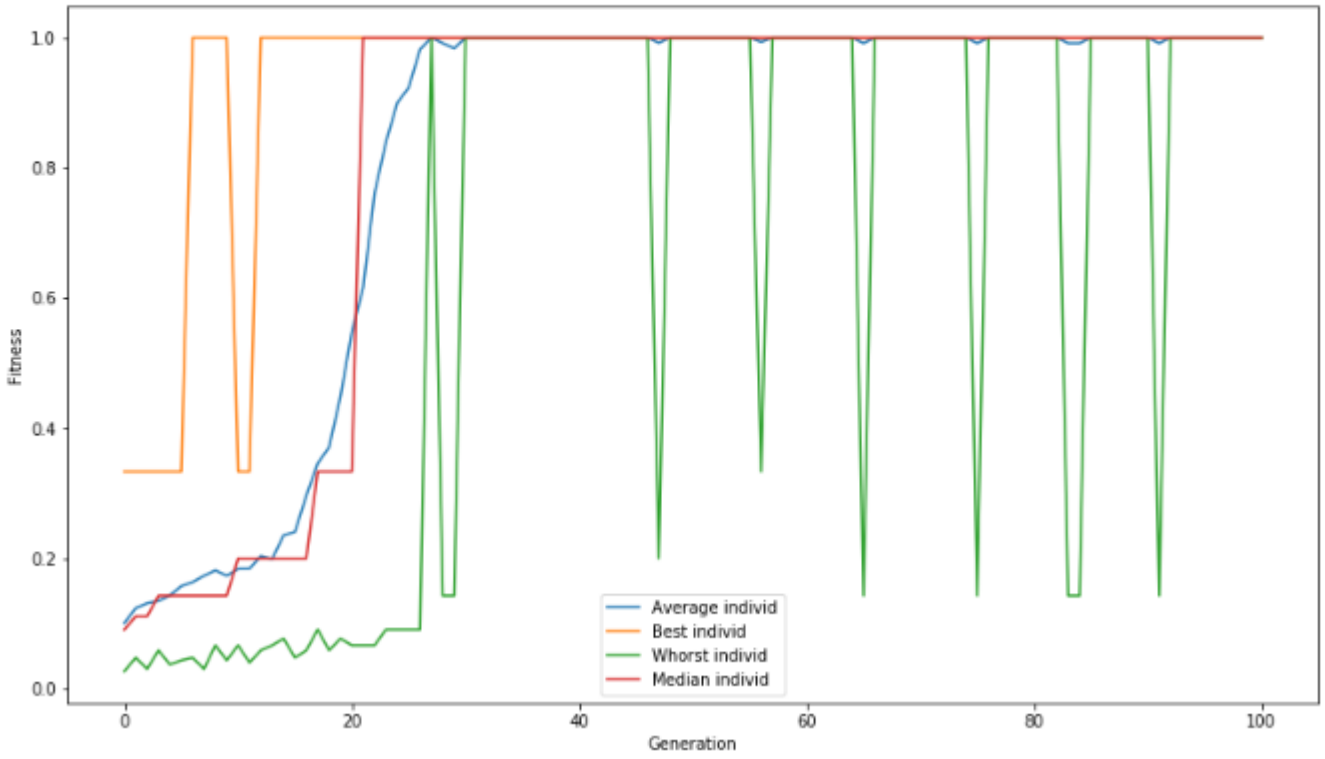
Permutation
Mutation probability 0.1%
Crossover probability 40%

PMX crossover
Swap mutation
Tournament Selection

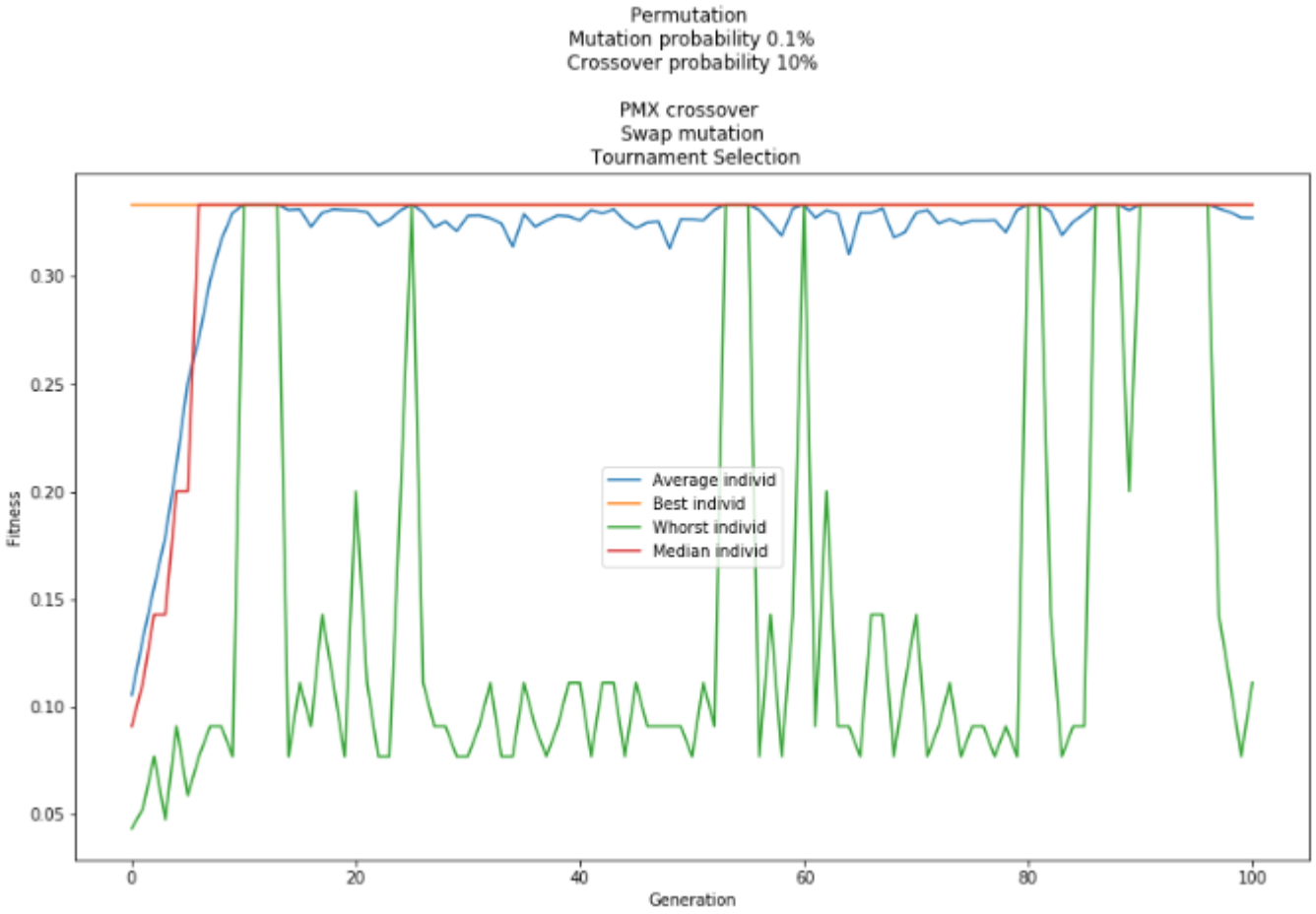


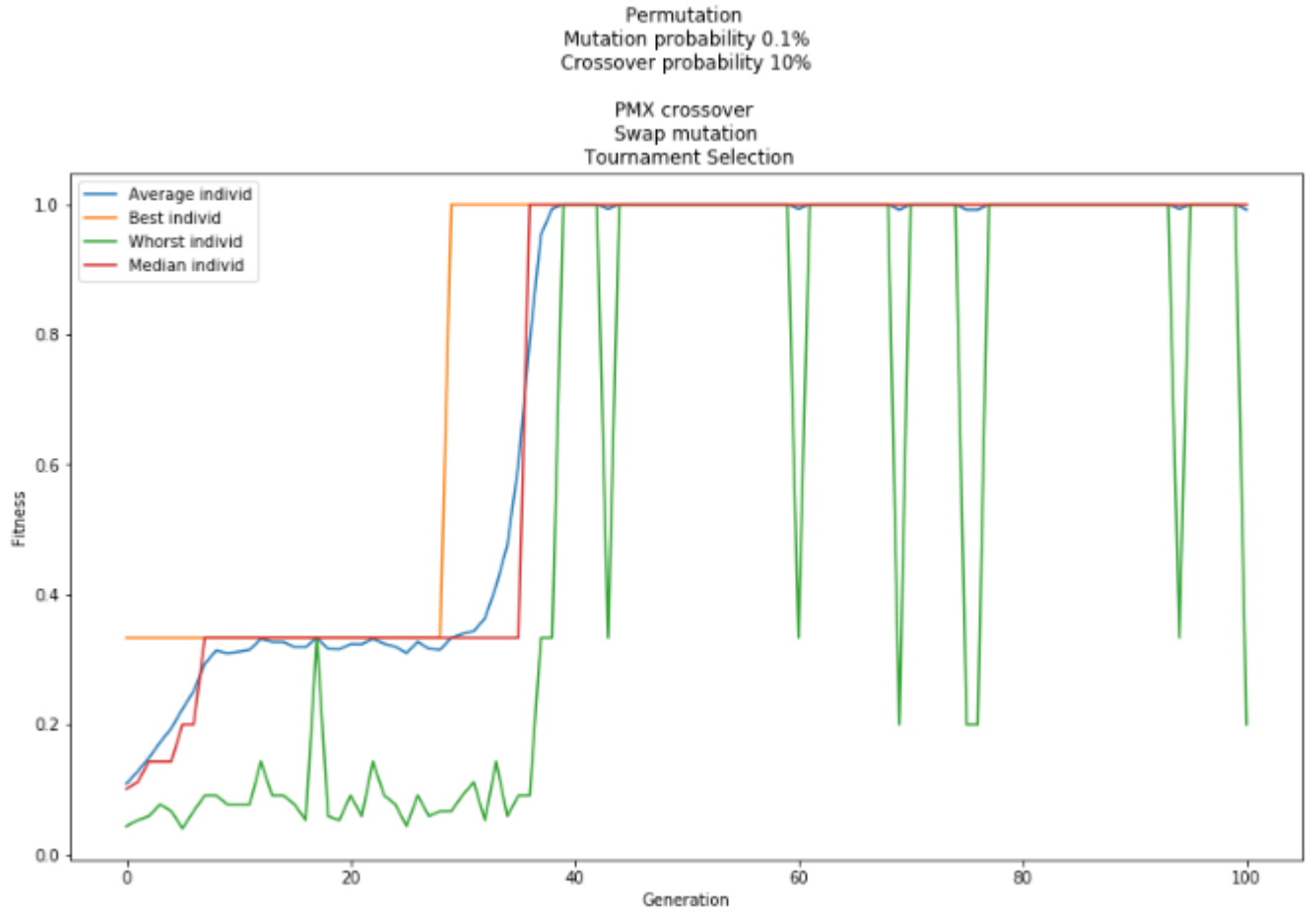
Permutation
Mutation probability 0.1%
Crossover probability 40%

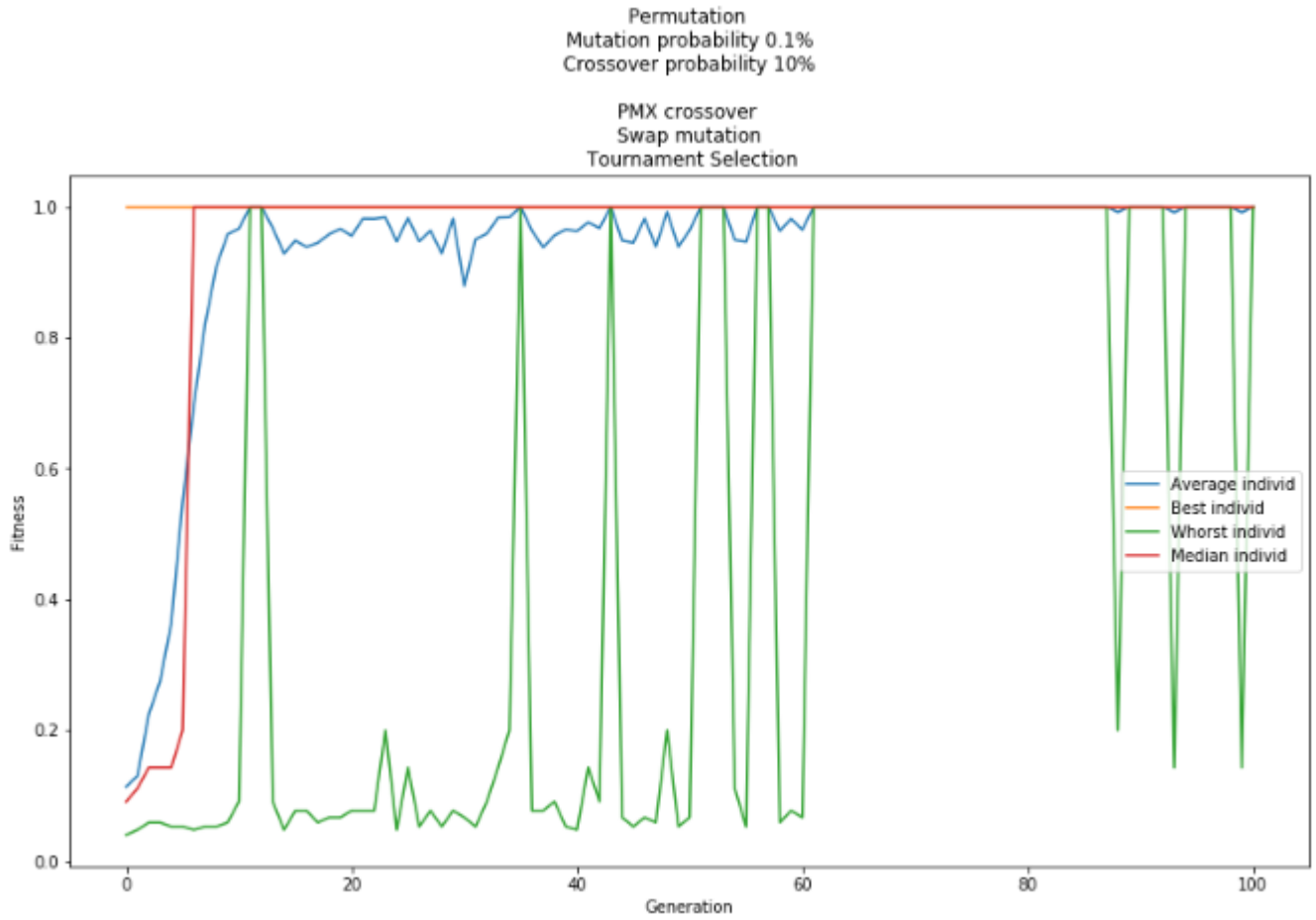
PMX crossover
Swap mutation
Tournament Selection



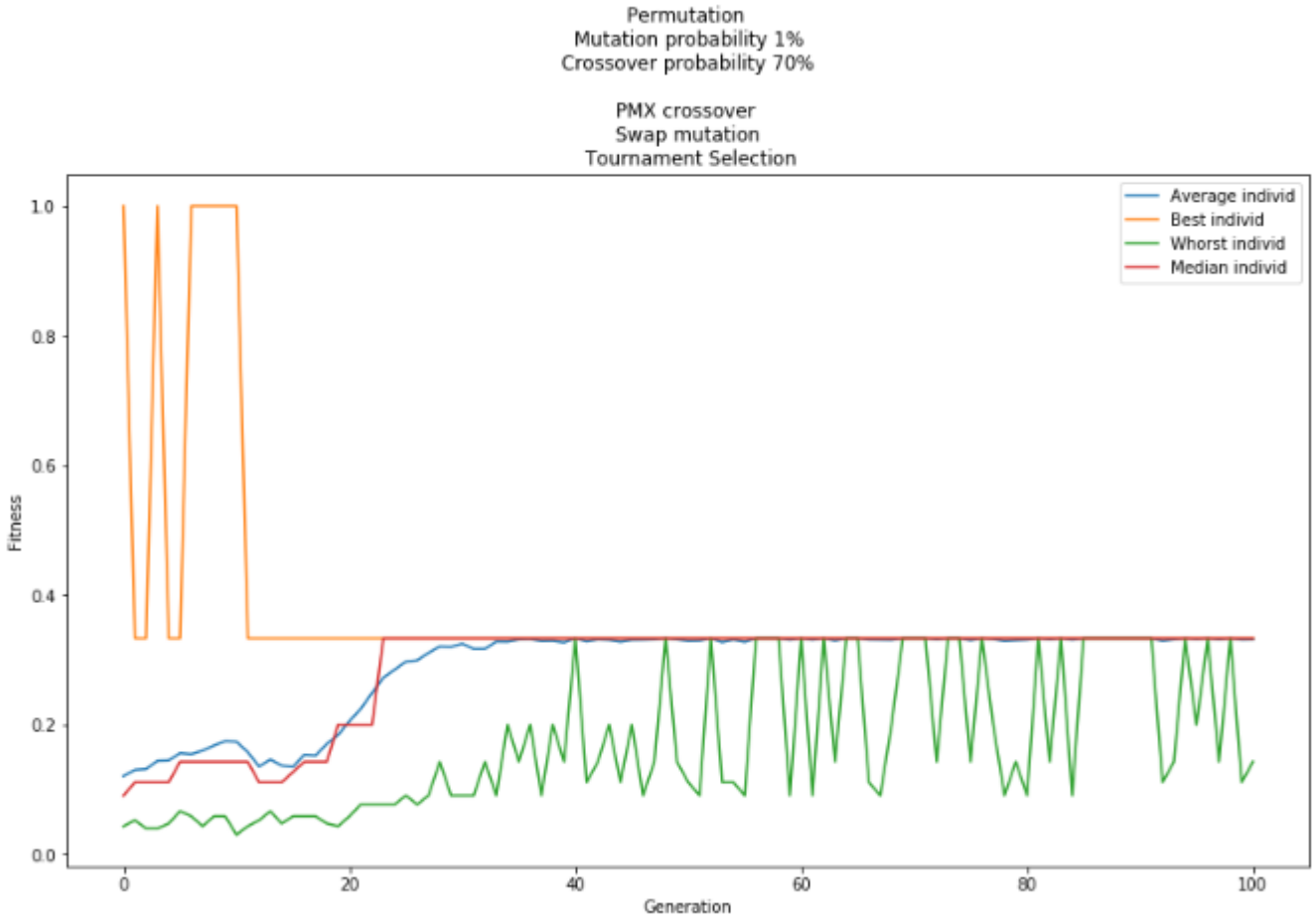
III. Running configuration with probabilities p_c=0.1, p_m=0.001:
Average time for termination: 0.37s





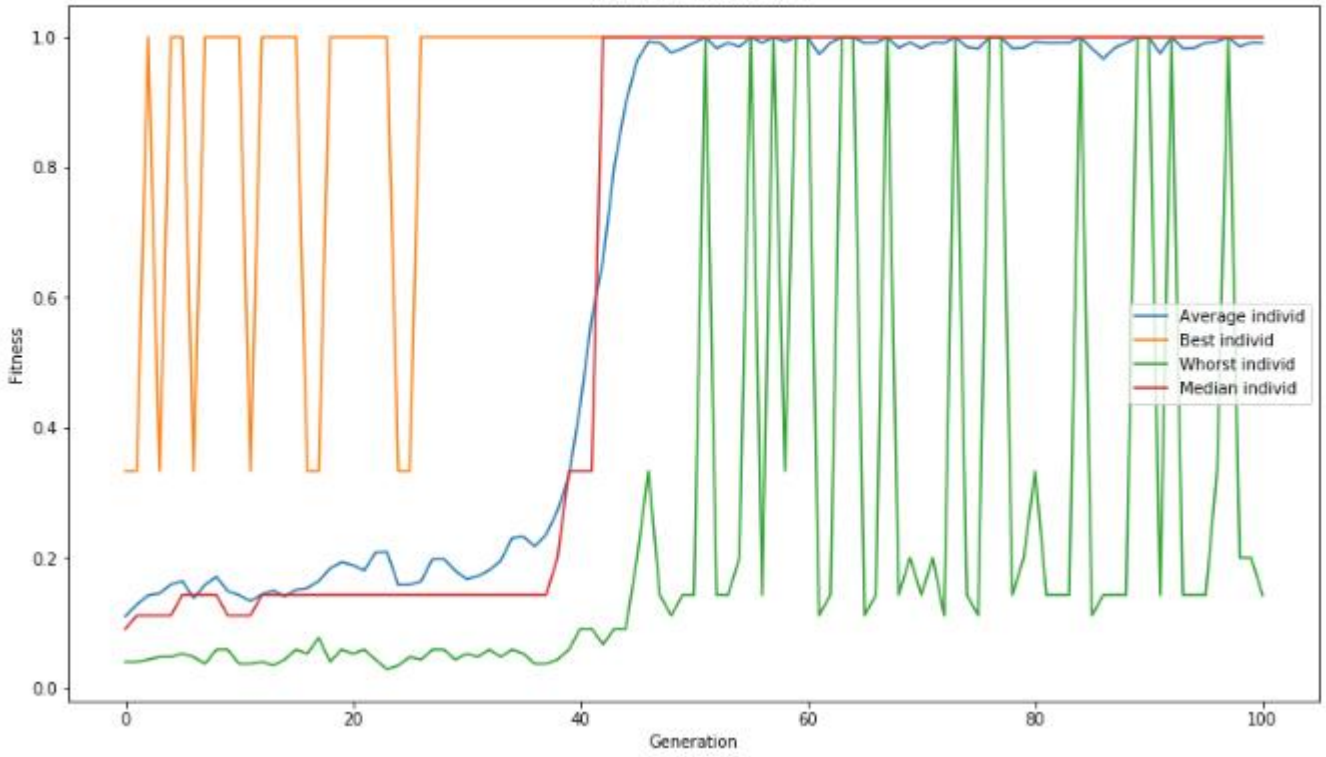


IV. Running configuration with probabilities p_c=0.7, p_m=0.01:
Average time for termination: 0.51s



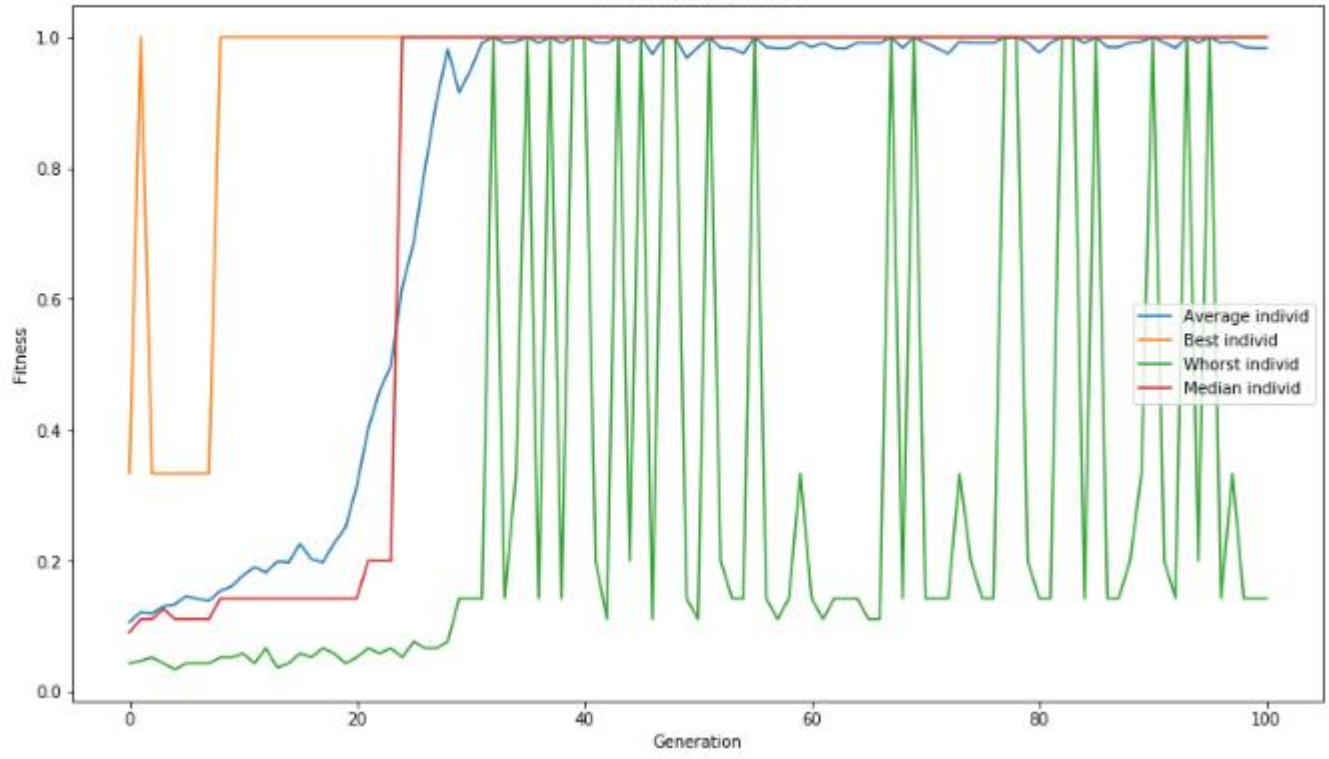
Permutation
Mutation probability 1%
Crossover probability 70%

PMX crossover
Swap mutation
Tournament Selection

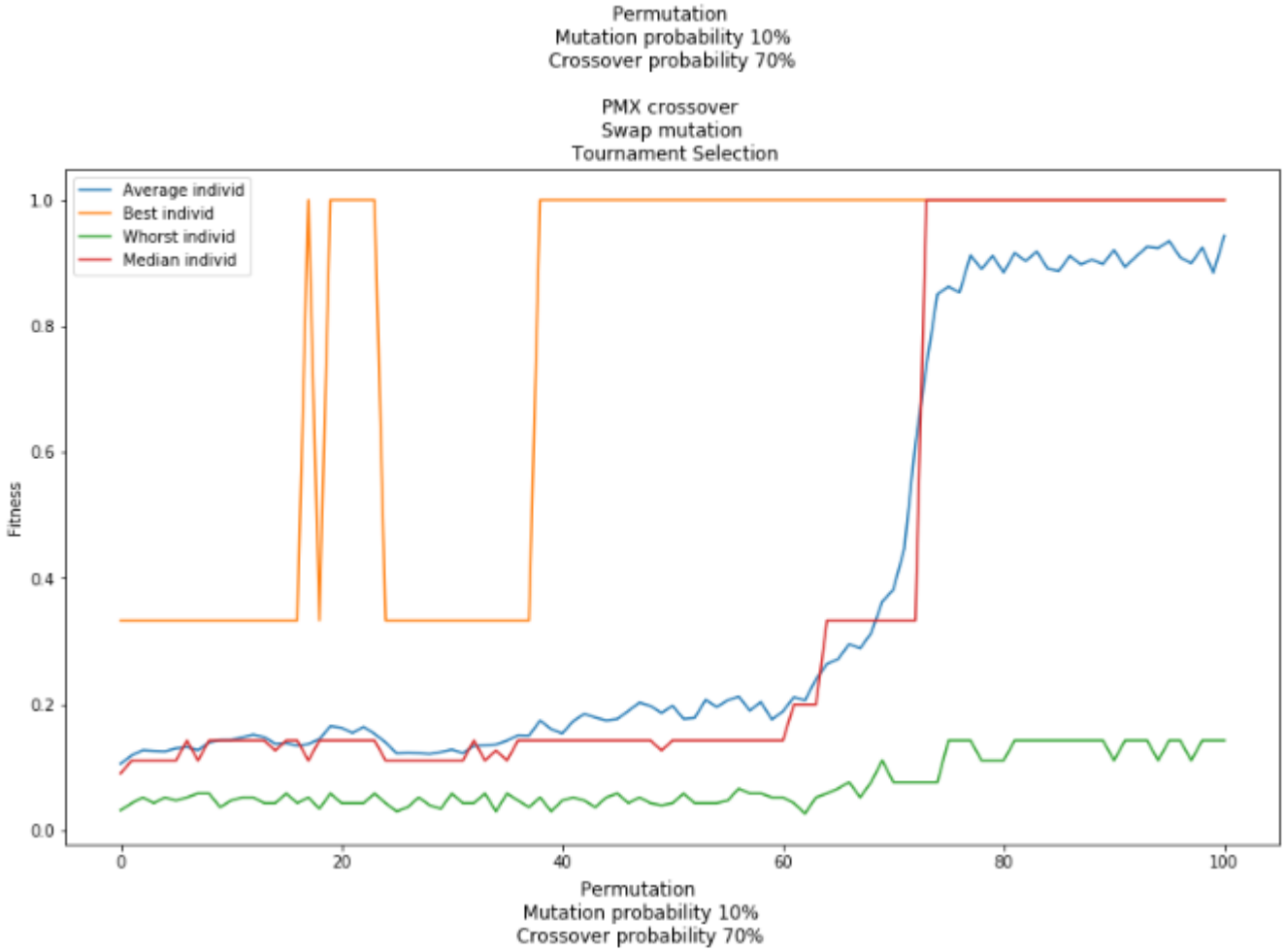


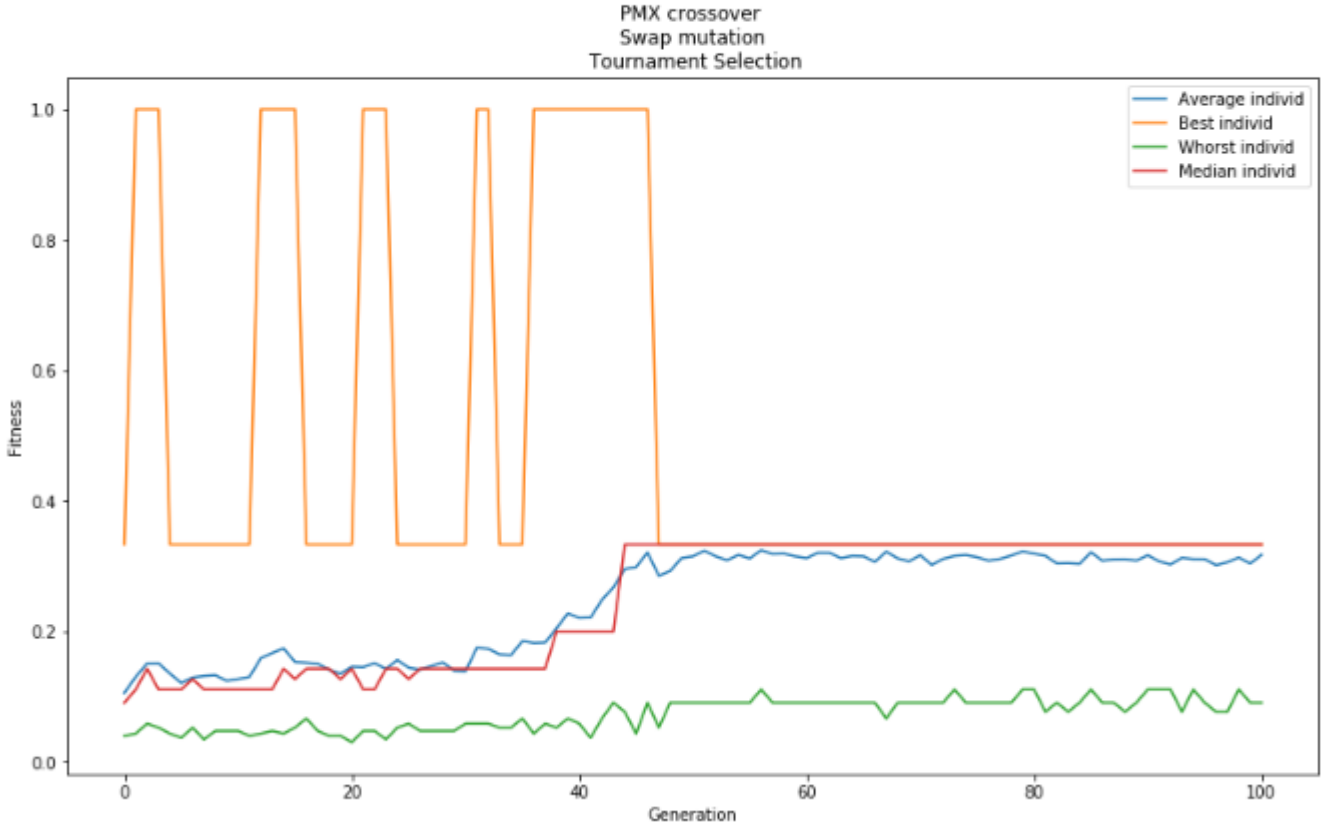
Permutation
Mutation probability 1%
Crossover probability 70%

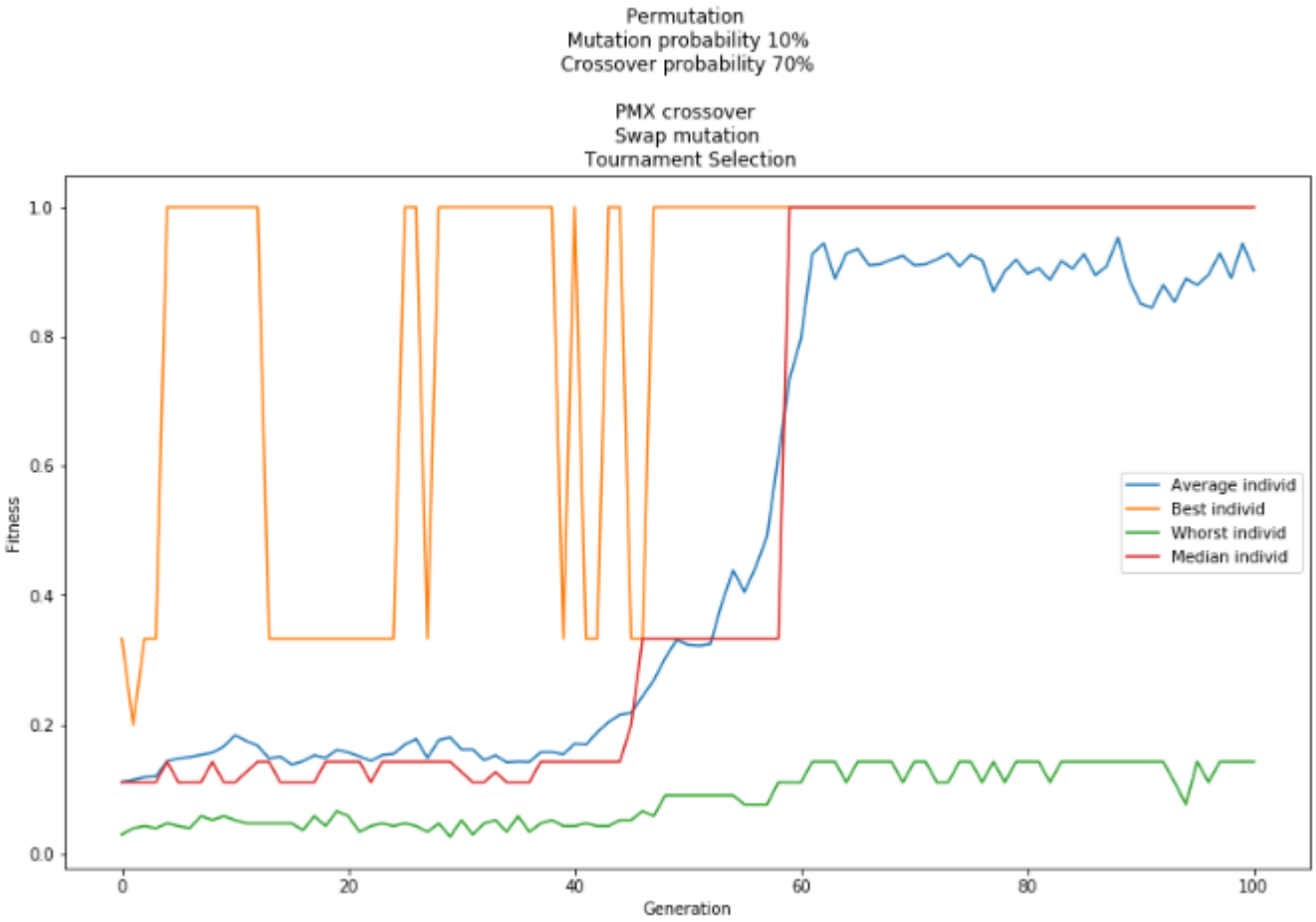
PMX crossover
Swap mutation
Tournament Selection



V. Running configuration with probabilities p_c=0.7, p_m=0.1:
Average time for termination: 0.51s







- In almost all configurations, we have reached the optimal solution.
- Larger changes (mutation, crossover) lead to a greater diversity of the population, as a result of which the population as a whole dwells less on a specific fitness value.
- With an increase in the probability of mutations, we see an increase in the scatter of fitness function values over all generations. This can lead to a wrong decision when the population is close to the right decision. It is this example that we see in the first figure at $p_c = 0.7$, $p_m = 0.01$ and in the third figure at $p_c = 0.7$, $p_m = 0.1$. But also it allows you to come to the right decision in earlier generations.
- With a lower probability of crossover (lower than 40%), the majority of children are represented by clones of their parents. From our graphs it follows that when the population is close to the right decision, we see jumps in the average fitness value, which makes it difficult for us to reach a global optimum. To achieve our goal and improve learning, we need to set a higher crossover probability.

Conclusions:

- The DEAP library turned out to be a convenient tool. Almost all the methods of variation operators we need are present there. The missing is easy to implement on your own.
- The evolutionary algorithm is a powerful enough tool to solve the eight queen's problem. However, for each specific task, it is necessary to select the appropriate operators parameters
- Representation of the genotype has a great influence. To solve the eight queen's problem with evolutionary algorithm, it is the representation in the form of a permutation that often leads to much more correct solutions.
- When comparing the average time for termination, in the representation in the form of permutations, it is two times less less than in the array representation.

- We also came to significant differences at different values of the crossover and mutation probabilities. In our experiments, the best configuration is: $p_c = 0.7$, $p_m = 0.001$ (in the second experiment).

Future Work:

- Optimization of the parameters of operators to achieve a global maximum with the least number of generations, respectively, and the time required to solve problems of this kind.

Credits:

- DEAP documentation https://deap.readthedocs.io/en/master/
- Sheppard Clinton. Genetic Algorithms with Python
- Eiben A.E., Smith J.E. Introduction to Evolutionary Computing