REPORT

Genetic Algorithms (GA)

Natural Selection Process Mimicking Biological Evolution

Abstract

John Holland gave a genetic Algorithm from the USA. It is an abstraction of natural biological evolution, like how parents give birth to children or how children evolve from their parents. However, it is used to solve complex problems rather than simpler ones. Additionally, it focuses on optimization utilizing a population of possible solutions for a given situation. Consequently, the best solution will survive. Thereby, a Genetic Algorithm (GA) is a search-based optimization technique based on Genetics and Natural Selection principles. Nevertheless, using this algorithm, we can find optimal solutions to complex problems which otherwise it would take a lifetime to solve.

Introduction

This report focuses on the tuning of hyperparameters to observe the effect on the performance of the Genetic Algorithm (GA). Thereby, we will be tuning a list of hyperparameters in this report to analyze their effect, namely:

- Mutation rate
- Crossover probability
- Number of Parents Mating
- K_Tournament size

The selection of fittest individuals from a ‘population’ leads to the start of the process of natural ‘selection’. Each individual has a ‘fitness value’ generated by a ‘fitness function’. Best individuals are selected on the basis of their fitness value. These individuals are called ‘parents’. They produce ‘offspring’ which inherits the characteristics of their parents, and they, in turn, will be added to the next ‘generation’. The offspring may carry ‘genes’ from both parents and this is called ‘crossover’. Conversely, the offspring

can carry a gene which does not exist in its parents, this is called 'mutation'. In addition to this, the offspring will be better than the parents and have a better chance of surviving if the parents have better fitness. However, the process continues until, in the end, we find a generation with the fittest individuals.

Methodology

Problem of Interest

In this report, research was conducted to analyze the performance of Genetic Algorithm (GA) while toggling different parameters.

Environment and Libraries

In order to make the research possible, Python programming language was used along with following python libraries and packages:

- Numpy
- PyGad
- Random
- Juypter Notebook

Flow of Work

Here is the summary of required steps:

1. Preparing the fitness_func.
2. Preparing Other Parameters.
3. Create an Instance of the **pygad.GA** Class.
4. Running the Genetic Algorithm.
5. Plotting Results.
6. Information about the Best Solution.
7. Repeating steps 2-6 for different parameters and different values of parameters

Example Utilized

The Genetic Algorithm was used to optimize the following function:

$$y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + \cdots .. + w_nx_n$$

$$where\ n = 32$$

In our example, the genotype consists of 32 genes also called 'weights'. The input to our equation contains decimal values instead of binary values:

function_inputs = [4, -2,3.5,5, -11, -4.7,4, -2,3.5,5, -11, -4.74, -2,3.5,5, -11, -4.74, -2,3.5,5, -11, -4.74, -2,3.5,5, -11, -4.74, -2,3.5,5, -11, -4.7]

The desired output value was set to:

$$y = 320$$

In order to achieve the desired output, the genetic algorithm is used to optimize the function to provide the best value for all 32 weights(w).

Fitness Function

The following fitness function was used to asses how close the solution is to the desired output. The higher the fitness value is, the closer is our solution to the desired output.

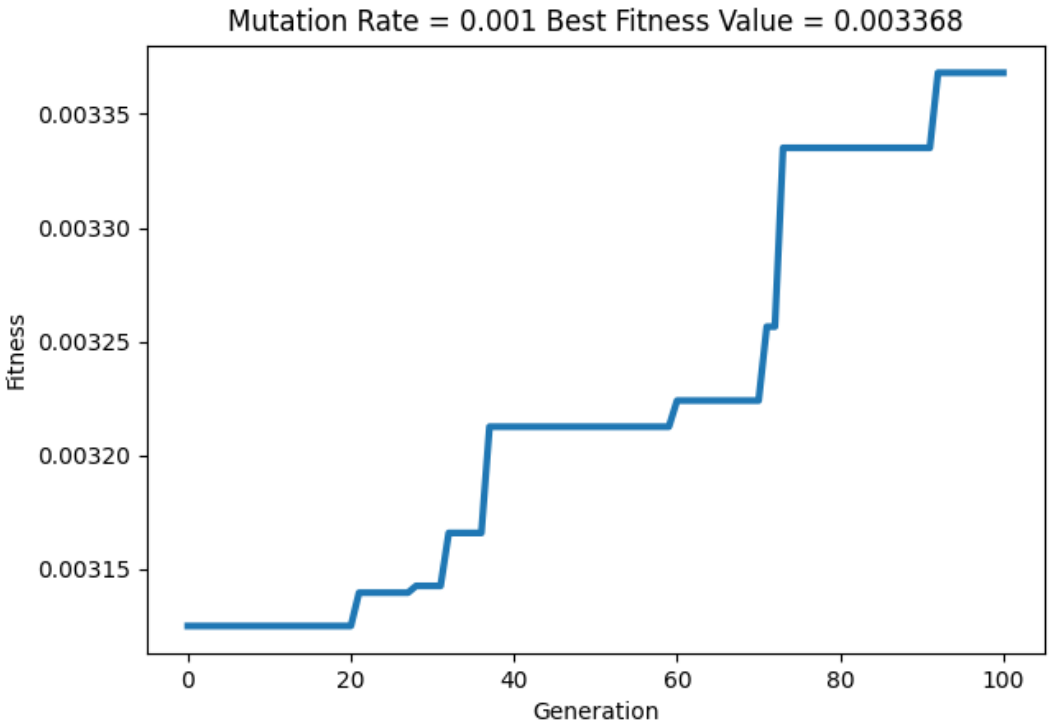
```
def fitness_func(solution, solution_idx):
    output = numpy.sum(solution*function_inputs)
    fitness = 1.0 / numpy.abs(output - desired_output)
    return fitness
```

Steps taken to ensure consistency of results by varying different parameters

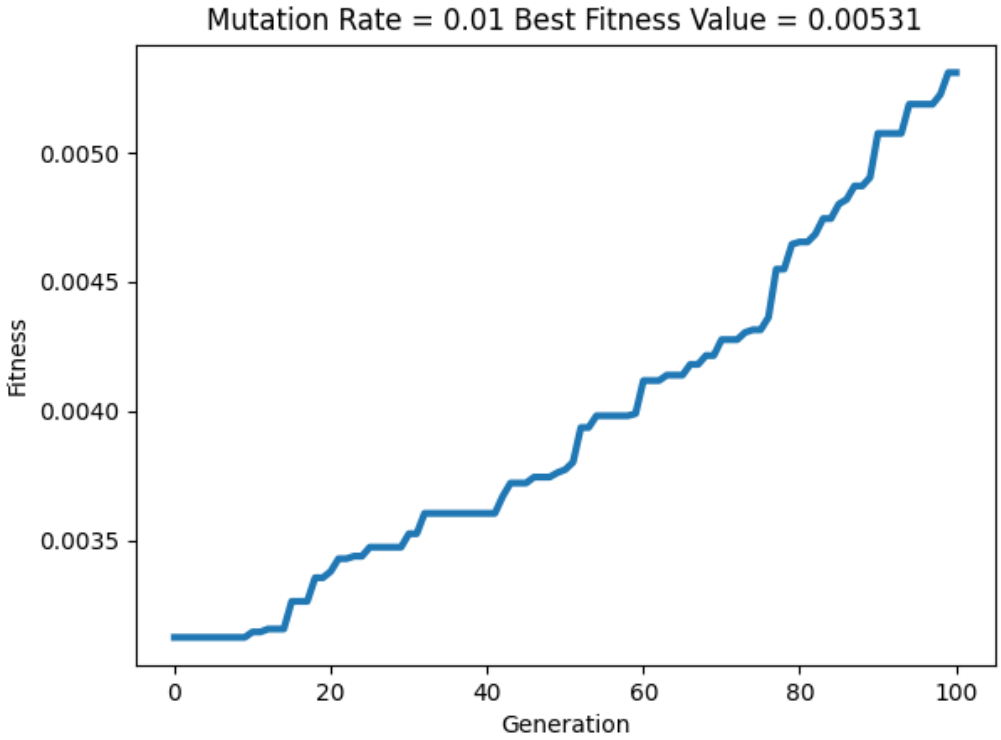
In order to avoid biasness and inconsistency in results, fixed number of generations were used in all experiments. Fixed value of initial population was used. Methods like random.seed(42) and numpy.random.seed(42) were used to generate same random numbers for all experiments. Number of solutions per population along with mutation type, crossover type, number of genes and fitness function were kept same for all experiments.

Results

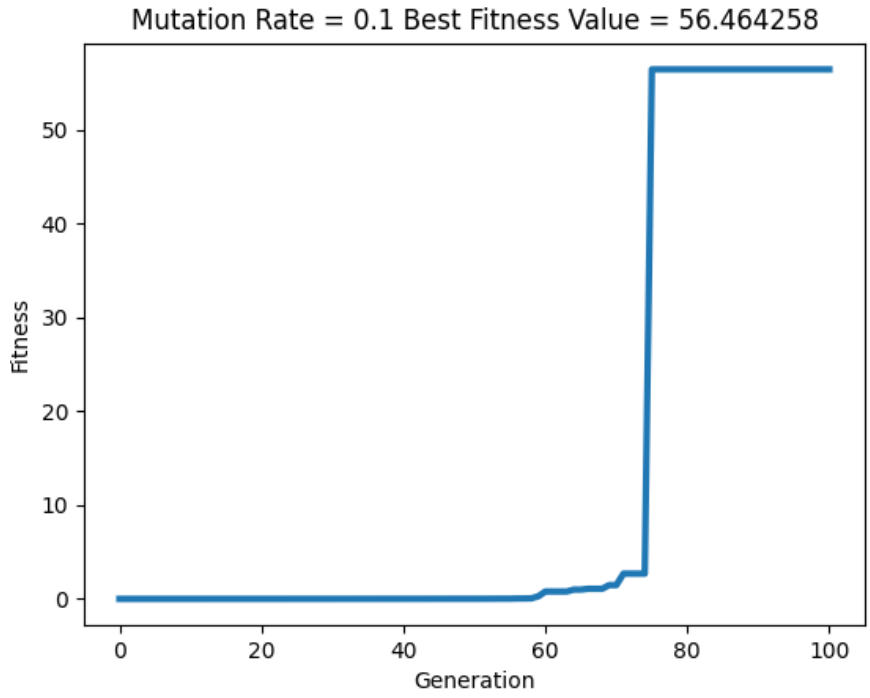
Effect of Mutation Rate on Performance while keeping all other factors constant



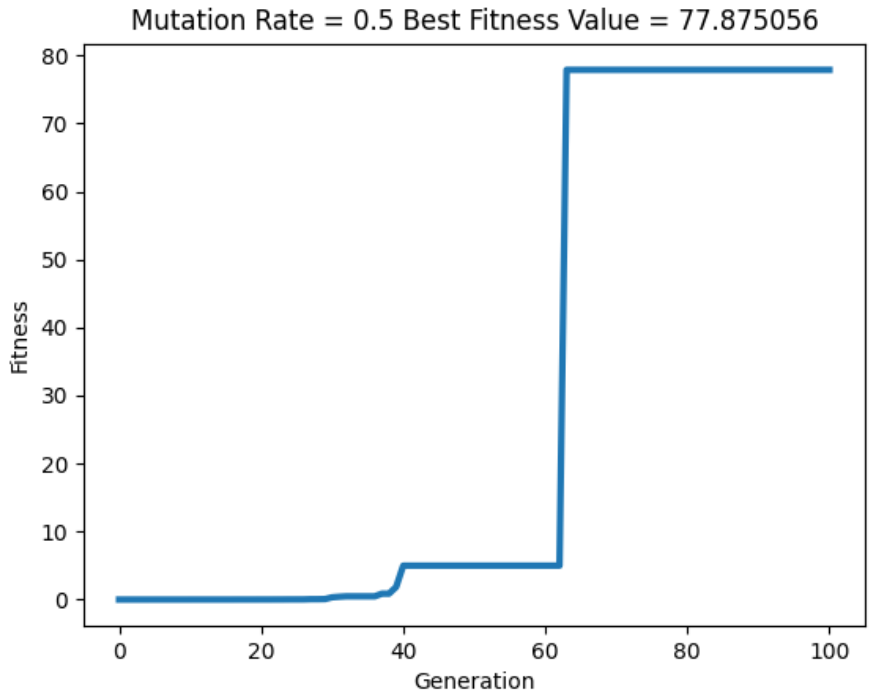
Generations	Mean	Standard deviation	Standard error
100	0.0032	8.4661	2.9655



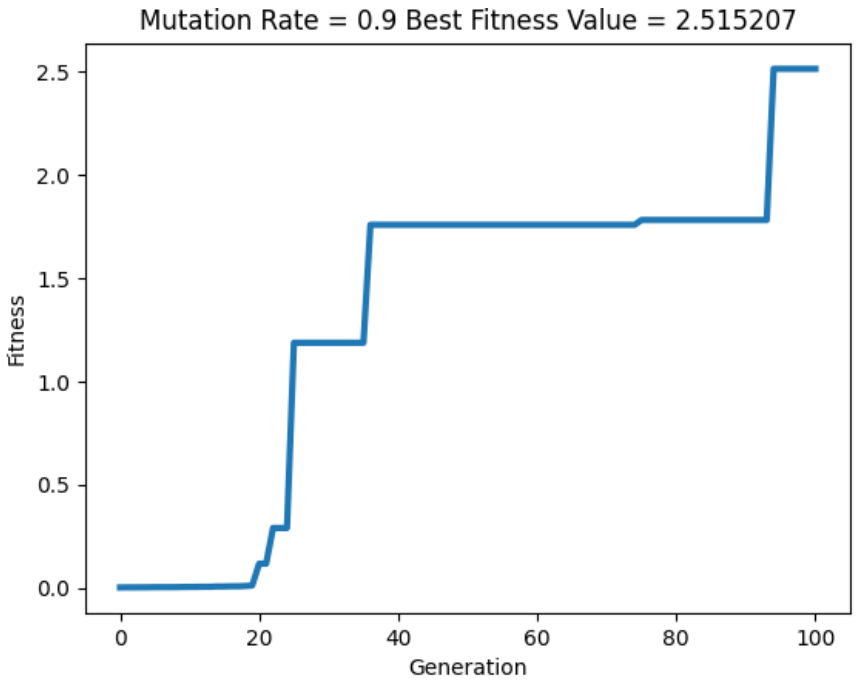
Generations	Mean	Standard deviation	Standard error
100	0.0039	0.0006	2.2313



Generations	Mean	Standard deviation	Standard error
100	2.9405	10.447	0.3659

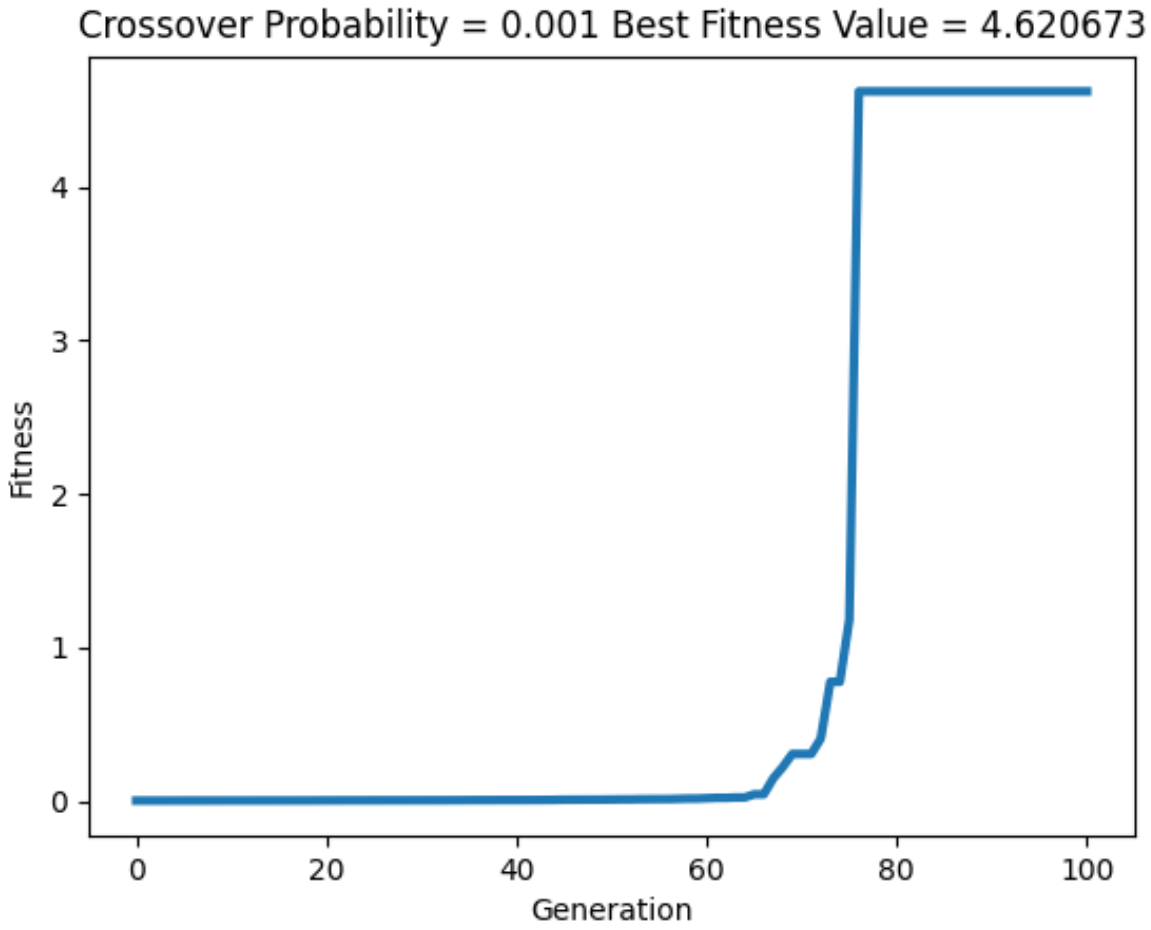


Generations	Mean	Standard deviation	Standard error
100	4.8534	16.5126	0.5784

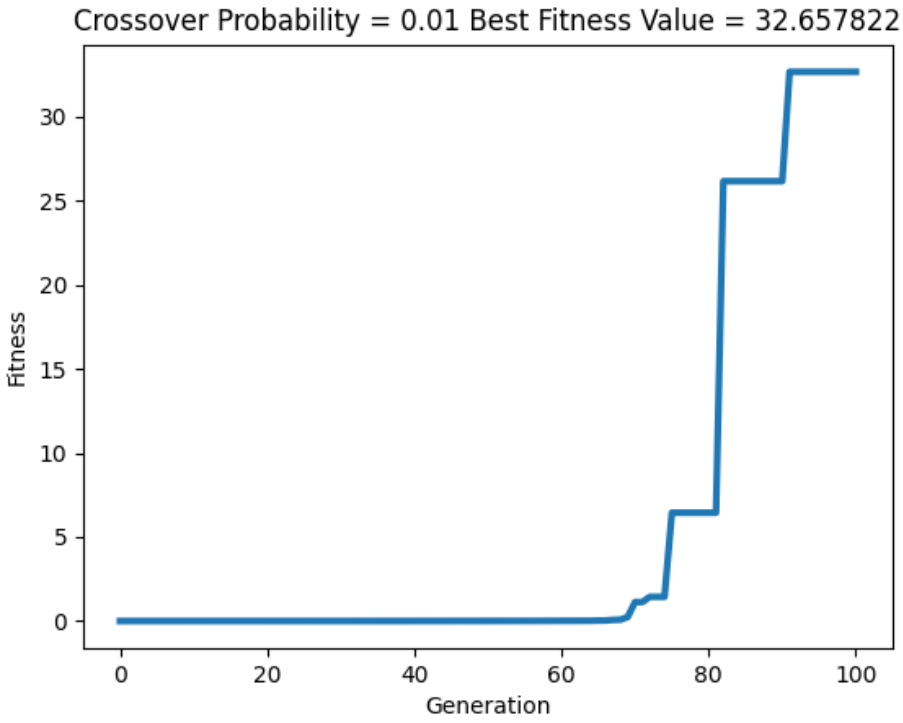


Generations	Mean	Standard deviation	Standard error
100	0.5485	0.6897	0.0241

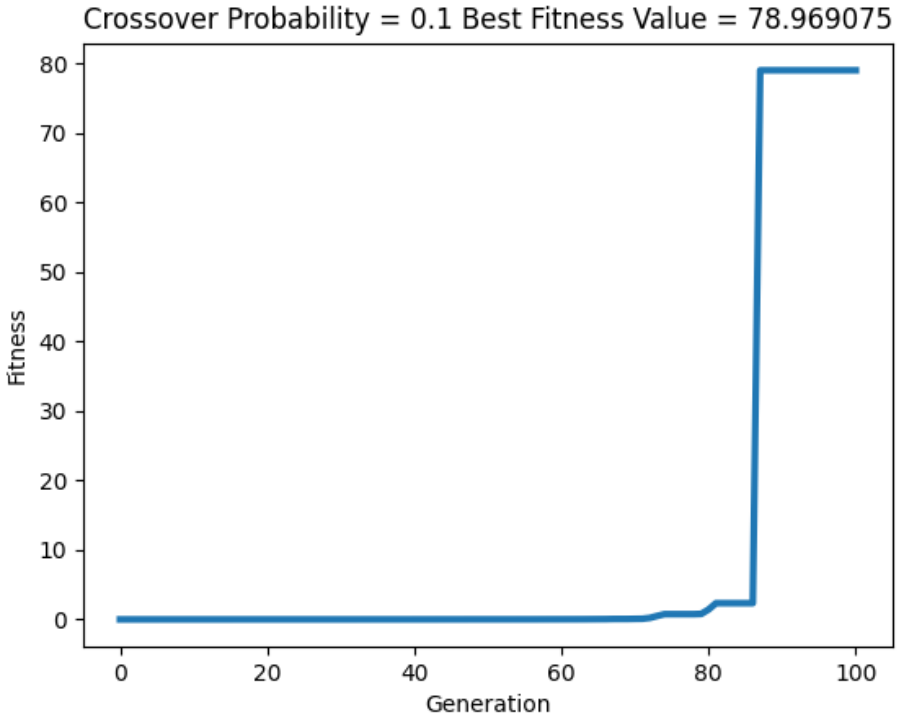
Effect of Crossover Probability on Performance while keeping all other factors constant



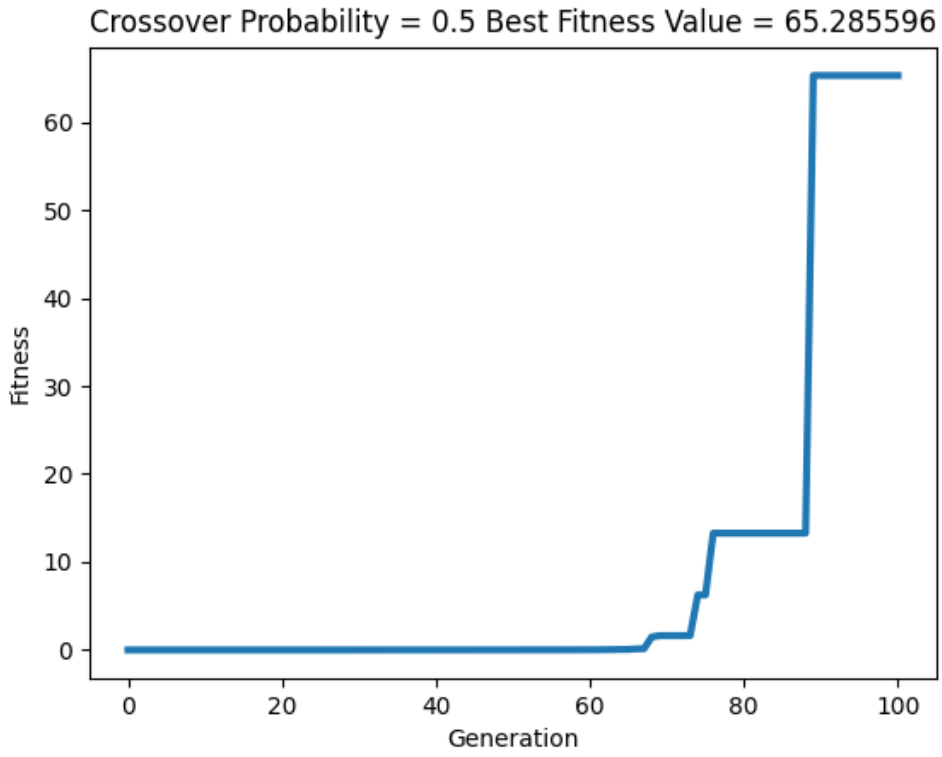
Generations	Mean	Standard deviation	Standard error
100	0.3966	0.9639	0.0337



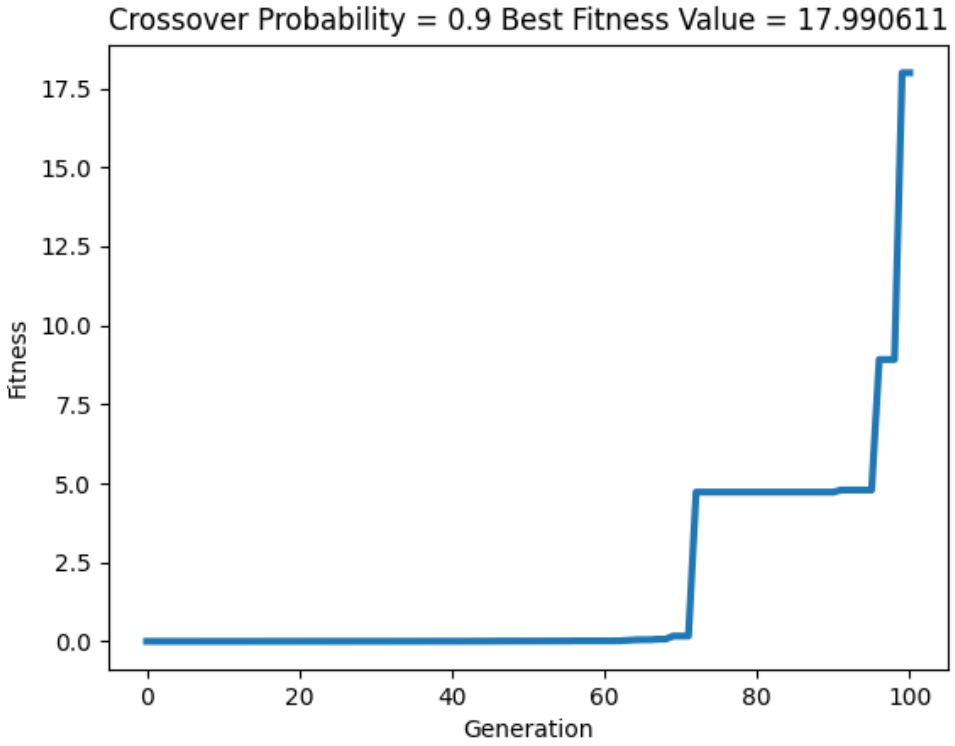
Generations	Mean	Standard deviation	Standard error
100	1.5258	5.5377	0.1939



Generations	Mean	Standard deviation	Standard error
100	1.7643	10.5989	0.3712

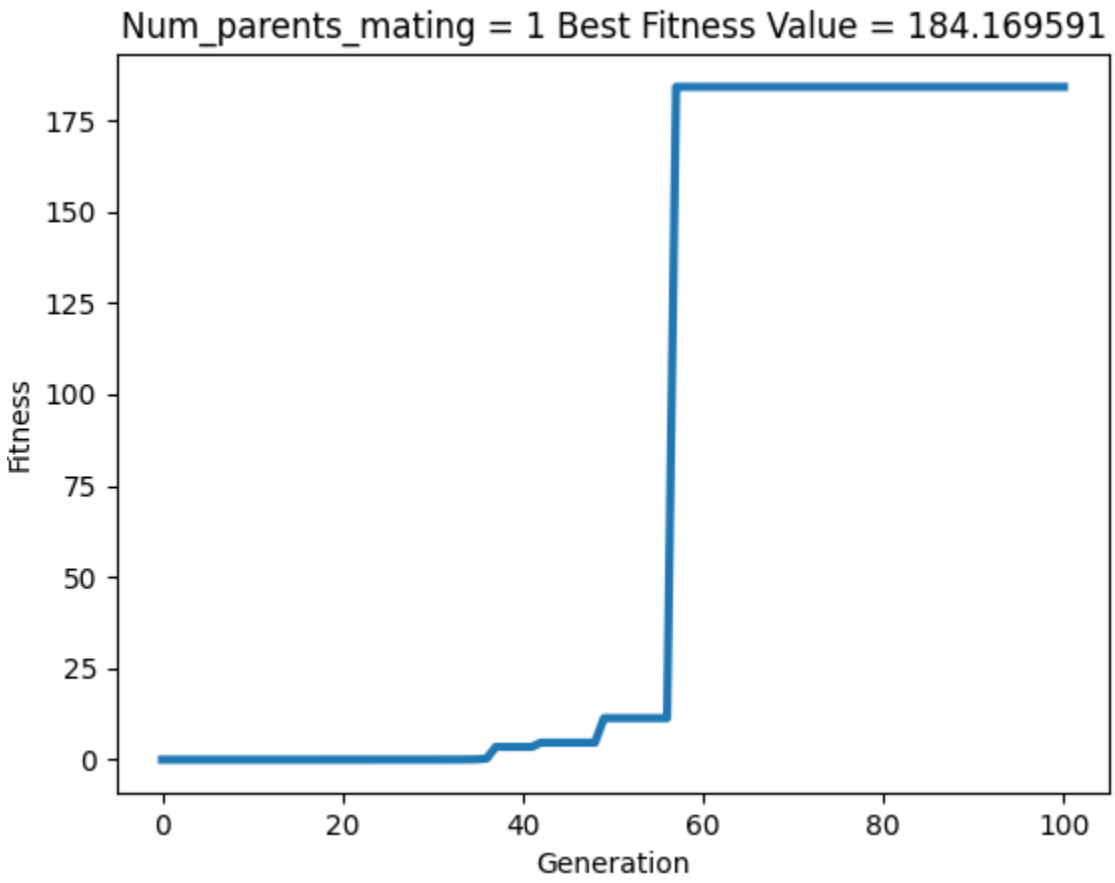


Generations	Mean	Standard deviation	Standard error
100	2.0658	8.5116	0.2981

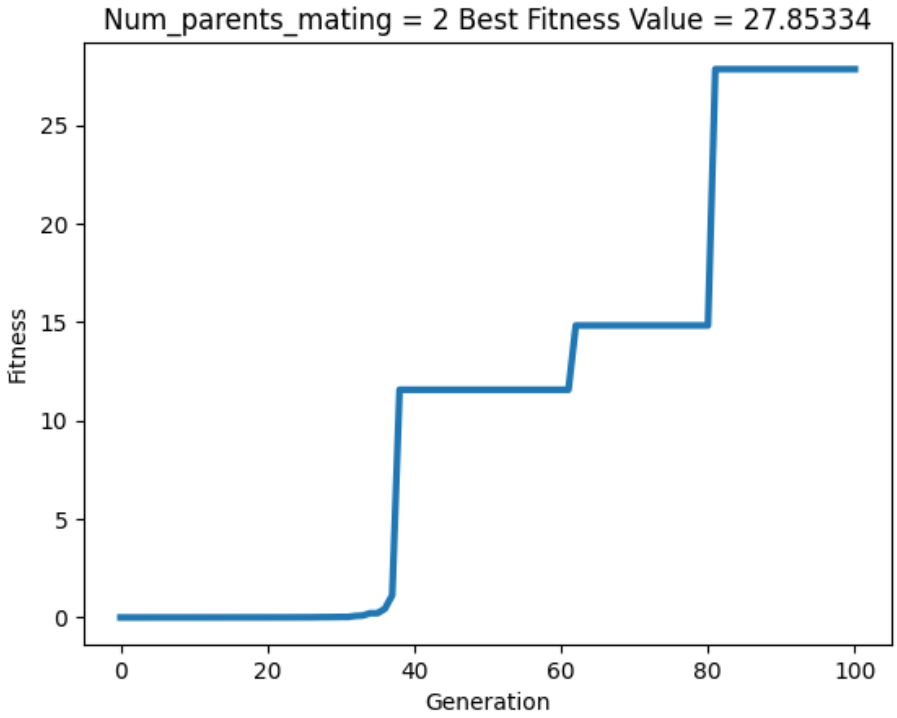


Generations	Mean	Standard deviation	Standard error
100	0.5725	1.7074	0.0598

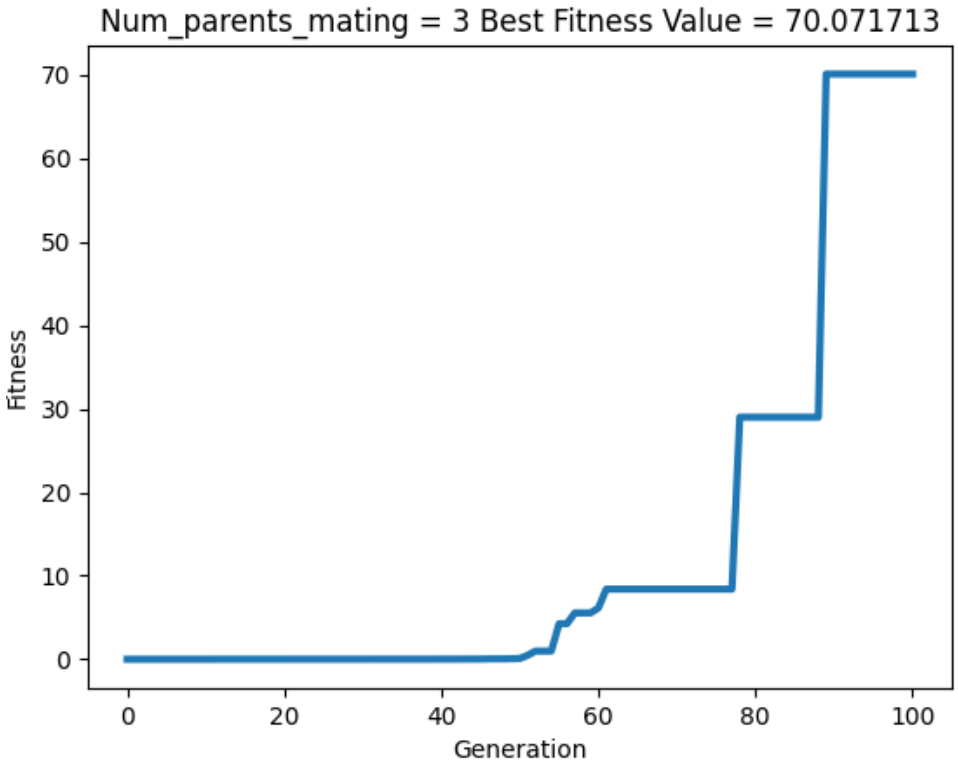
Effect of Num_parents_mating on Performance while keeping all other factors constant



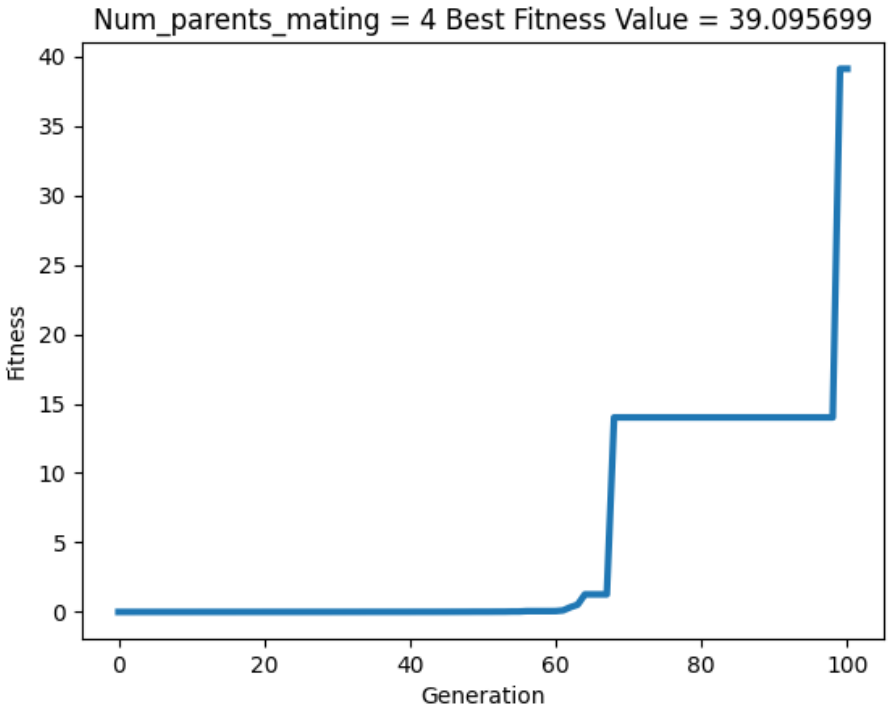
Generations	Mean	Standard deviation	Standard error
100	10.8624	41.9958	1.4710



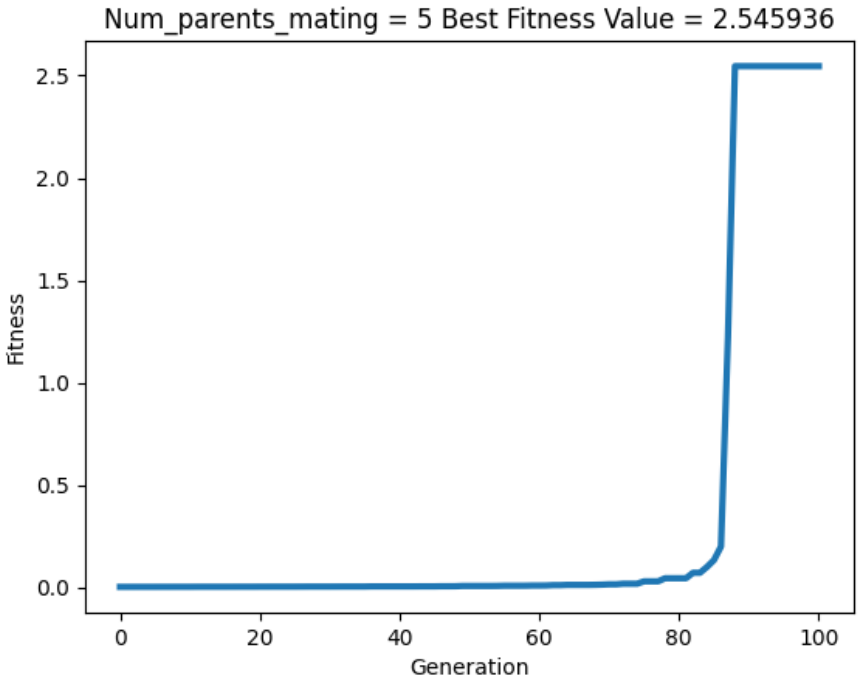
Generations	Mean	Standard deviation	Standard error
100	2.5986	5.7969	0.2030



Generations	Mean	Standard deviation	Standard error
100	3.4882	10.8572	0.3803

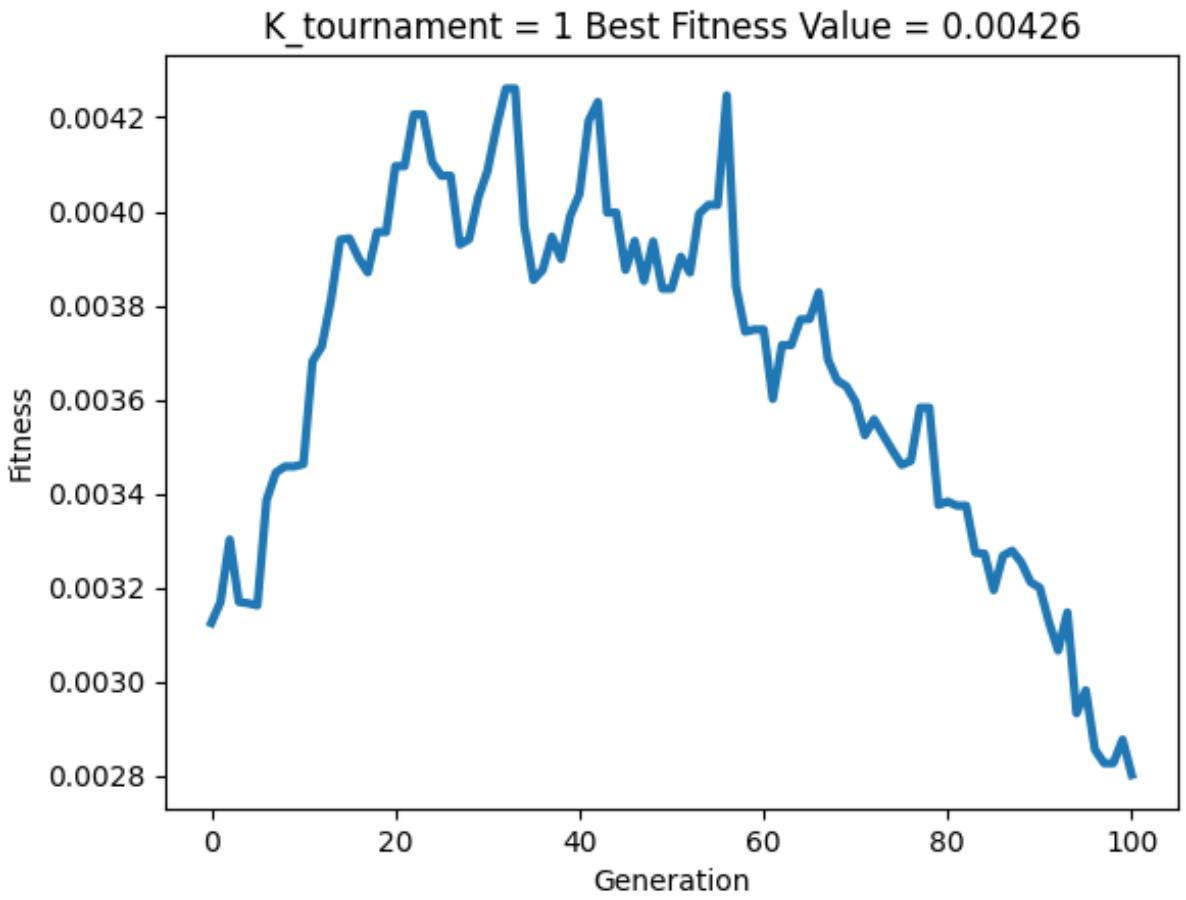


Generations	Mean	Standard deviation	Standard error
100	1.3153	4.2005	0.1471

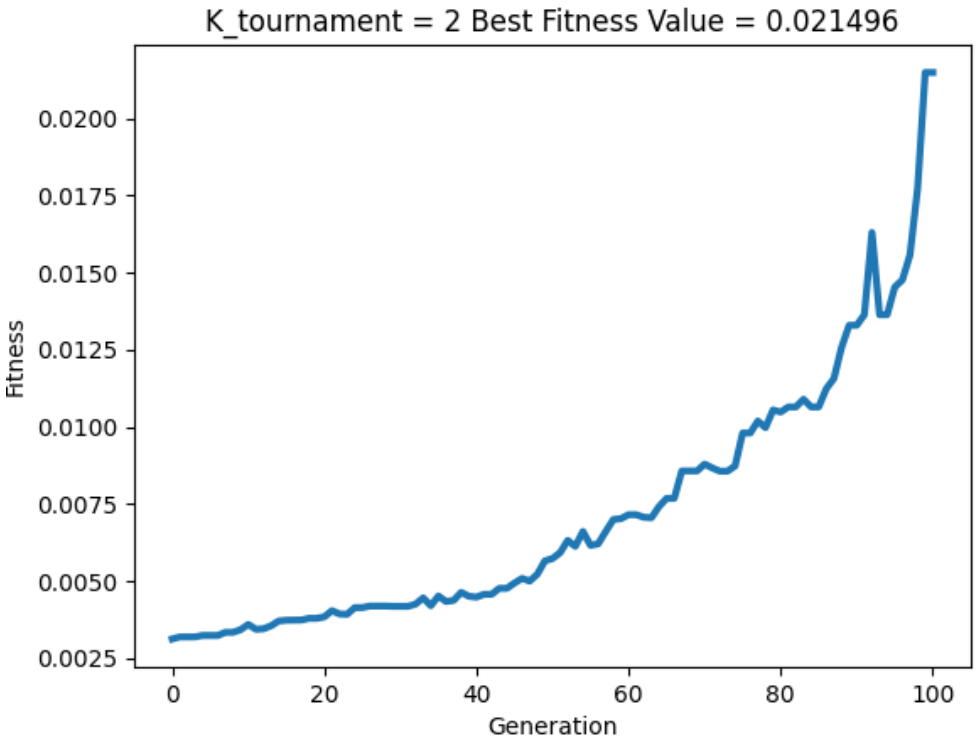


Generations	Mean	Standard deviation	Standard error
100	0.1228	0.3910	0.0136

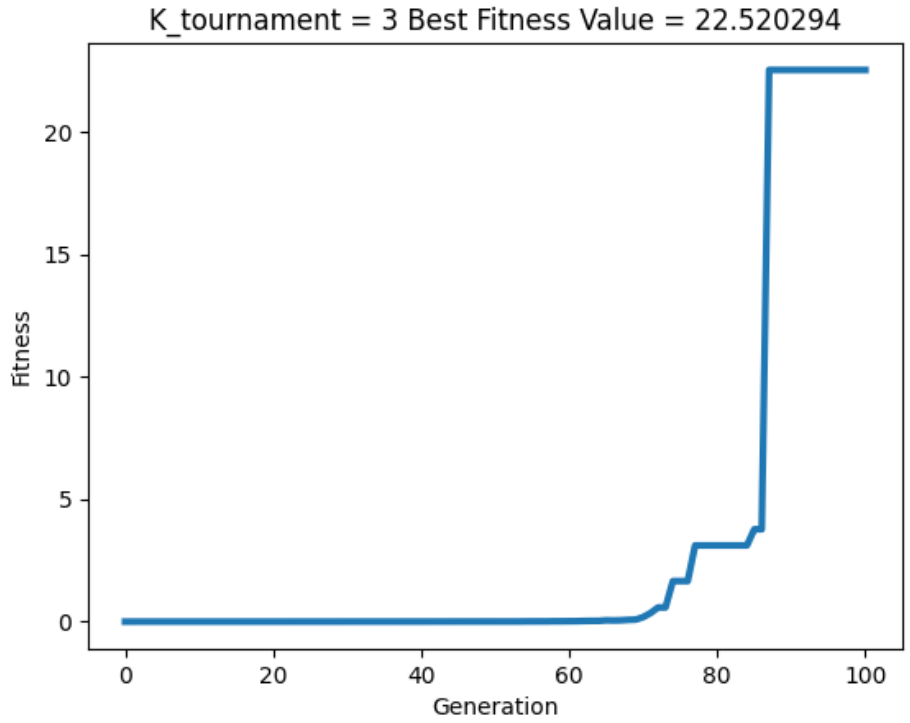
Effect of K_Tournament size on Performance while keeping all other factors constant



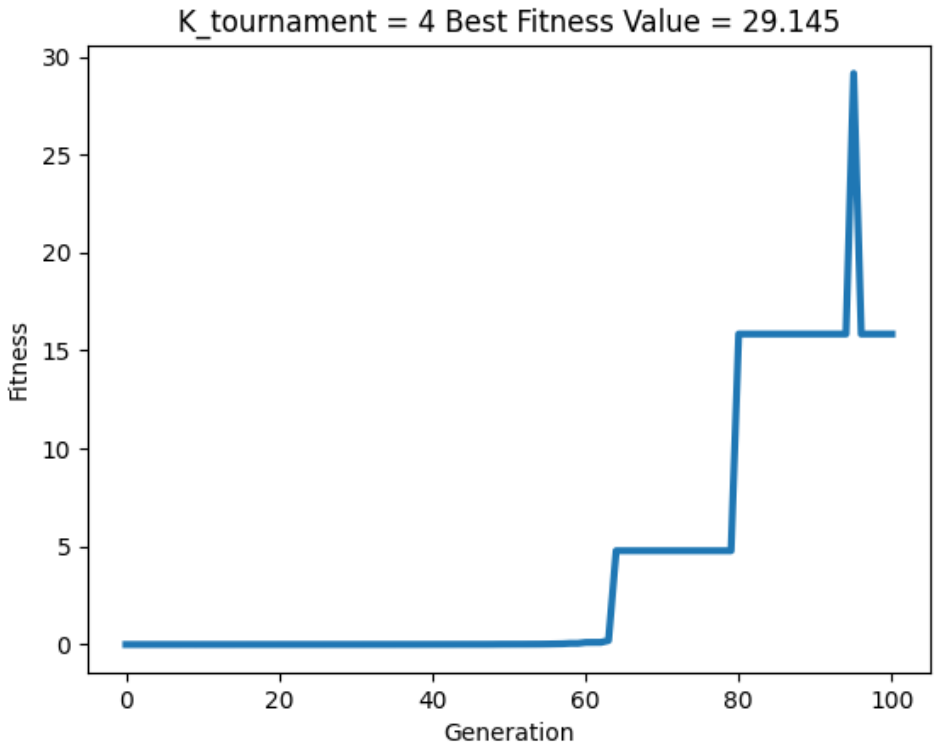
Generations	Mean	Standard deviation	Standard error
100	0.0035	0.0003	1.2776



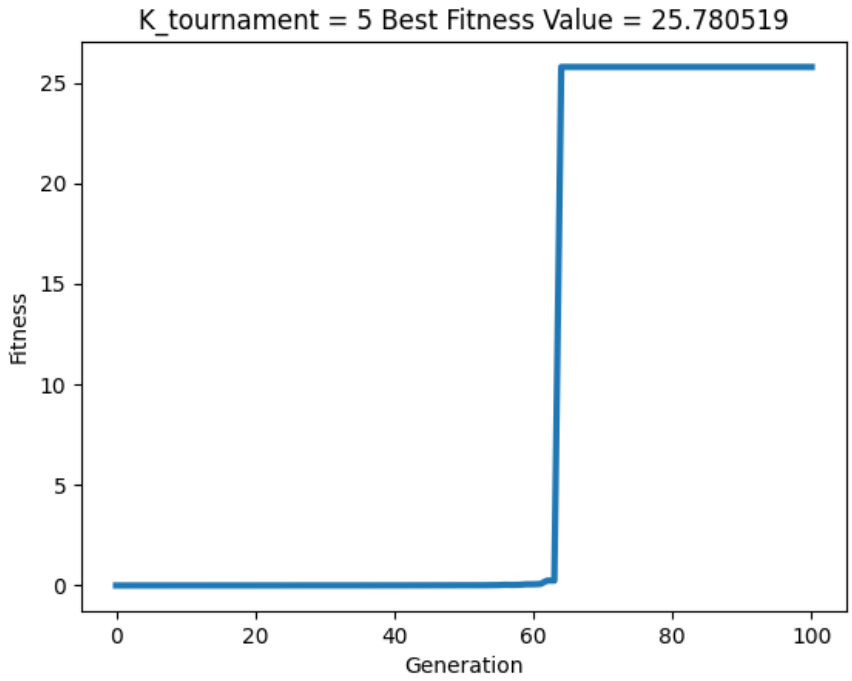
Generations	Mean	Standard deviation	Standard error
100	0.0067	0.0036	0.0001



Generations	Mean	Standard deviation	Standard error
100	1.5971	5.2830	0.1850



Generations	Mean	Standard deviation	Standard error
100	1.9537	4.6677	0.1635



Generations	Mean	Standard deviation	Standard error
100	4.5973	9.6891	0.3393

Discussion

Mutation Rate

It is the probability of selecting a gene for applying the mutation operation. Every gene in the solution is assigned a random value between 0 and 1. If this random value is less than or equal to the value assigned to the mutation rate parameter, then the gene is selected.

From the results above, as mutation rate was increased, the fitness value also increased. However, the last figure depicts that if we increase mutation rate above a certain threshold, then the performance of genetic algorithm drastically decreases.

Crossover Probability

It is the probability of selecting a parent for applying the crossover operation. A random value is assigned to each parent and similar to mutation rate, its value is between 0 and 1. If this random value is less than or equal to the value assigned to the crossover probability parameter, then the parent is selected.

The effect of crossover probability is quite similar to the mutation rate. If the probability is too low then the performance of Genetic Algorithm downgrades, moreover, if the value is too large, then also the performance of Genetic Algorithm decreases. Thus, an optimal value of crossover probability will result in increased performance.

Number of Parents Mating

This parameter value signifies the number of solutions to be selected as parents.

When tuning this hyperparameter, the best performance was achieved when its value was set to 1. The performance declined as we increased the number of parents mating.

K_tournament Size

In order to make use of this parameter, the parent selection type was changed from steady state selection to tournament selection. This parameter specifies the number of parents participating in the tournament selection.

The performance of Genetic Algorithm increased as value of K_tournament elevated. However, this parameter seemed trivial as performance declines when number of generations are high.

Conclusion

Conclusively, this study could be more effective if we analyze the effect of other parameters like, parent selection type, crossover operator type, mutation operator type and fitness function. Good results can be yielded if we try vast range of values for a specific parameter in order to discover its optimal value. Nevertheless, this study proved to be interesting while analyzing effects of different parameters on performance of Genetic Algorithm.

References

- Gad, A., 2021. *Genetic Algorithm Implementation in Python*. [online] towardsdatascience. Available at: <https://towardsdatascience.com/genetic-algorithm-implementation-in-python-5ab67bb124a6> [Accessed 20 May 2021].
- Gad, A., 2021. *Adaptive Mutation in Genetic Algorithm with Python Examples*. [online] neptune.ai. Available at: <https://neptune.ai/blog/adaptive-mutation-in-genetic-algorithm-with-python-examples> [Accessed 20 May 2021].
- Gad, A., 2020. *pygad Module — PyGAD 2.14.1 documentation*. [online] Pygad.readthedocs.io. Available at: <https://pygad.readthedocs.io/en/latest/README_pygad_ReadTheDocs.html> [Accessed 20 May 2021].
- Gad, A., 2021. *Adaptive Mutation in Genetic Algorithm with Python Examples*. [online] neptune.ai. Available at: <https://neptune.ai/blog/adaptive-mutation-in-genetic-algorithm-with-python-examples> [Accessed 20 May 2021].
- Gad, A., 2018. *Introduction to Optimization with Genetic Algorithm*. [online] towardsdatascience. Available at: <https://towardsdatascience.com/introduction-to-optimization-with-genetic-algorithm-2f5001d9964b> [Accessed 20 May 2021].