Coursework 1: Biologically inspired computing for optimization

By Dimo Dimchev

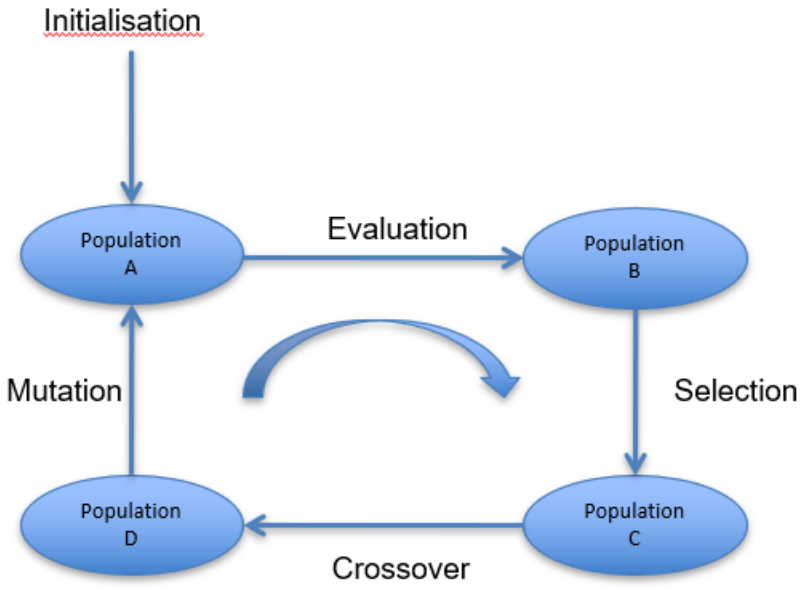
Intro

In this report I explain my approach for optimization for the two algorithms – GA and PSO.

The algorithms use a minimization function as a benchmark, meaning their best solution is 0.

Genetic algorithm description:

The algorithm reflects the process of natural selection where the best individuals survive and reproduce in order to produce offspring of the next generation.



The algorithm consists of 5 phases.

Initialization – generating the initial population. Every individual is characterized by a set of parameters called Genes. Genes are joined together to form a Chromosome.

Evaluation – done through a fitness formula that determines how fit the individual is and assigns a fitness score to it.

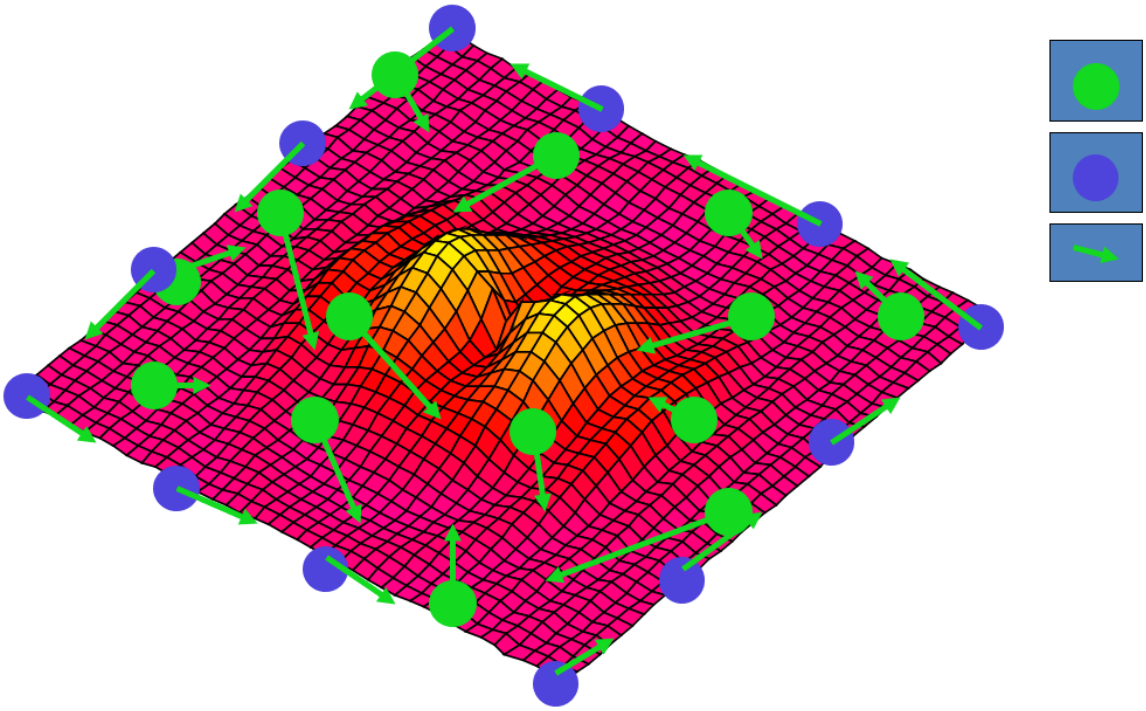
Selection – select the fittest individuals. An individual with high fitness has better chance of being selected for reproduction.

Crossover -individuals mate.

Mutation – a chance that some individual can mutate.Occurs to maintain diversity in the population.

Particle swarm optimization (PSO) description:

PSO is based on the research of bird and fish flock movement behaviors.



The PSO algorithm consists of **n** particles, and each particle is a potential solution in D-dimensional space. The particles move according to tree principles:

1.A particle keeps its inertia

2. A particle changes its condition according to its most optimal position

3. A particle changes its condition according to the global most optimal position.

My approach to optimization!

The plan that I had for optimizing the GA and PSO algorithms consisted of 4 steps.

Step 0: Run the algorithm with Default values and see how best fitness improves over time.

Step 1:Test every operator(change only one parameter at a time, while keeping all other parameters fixed) for Its whole range of values and see how it behaves.(For PSO just test all the)

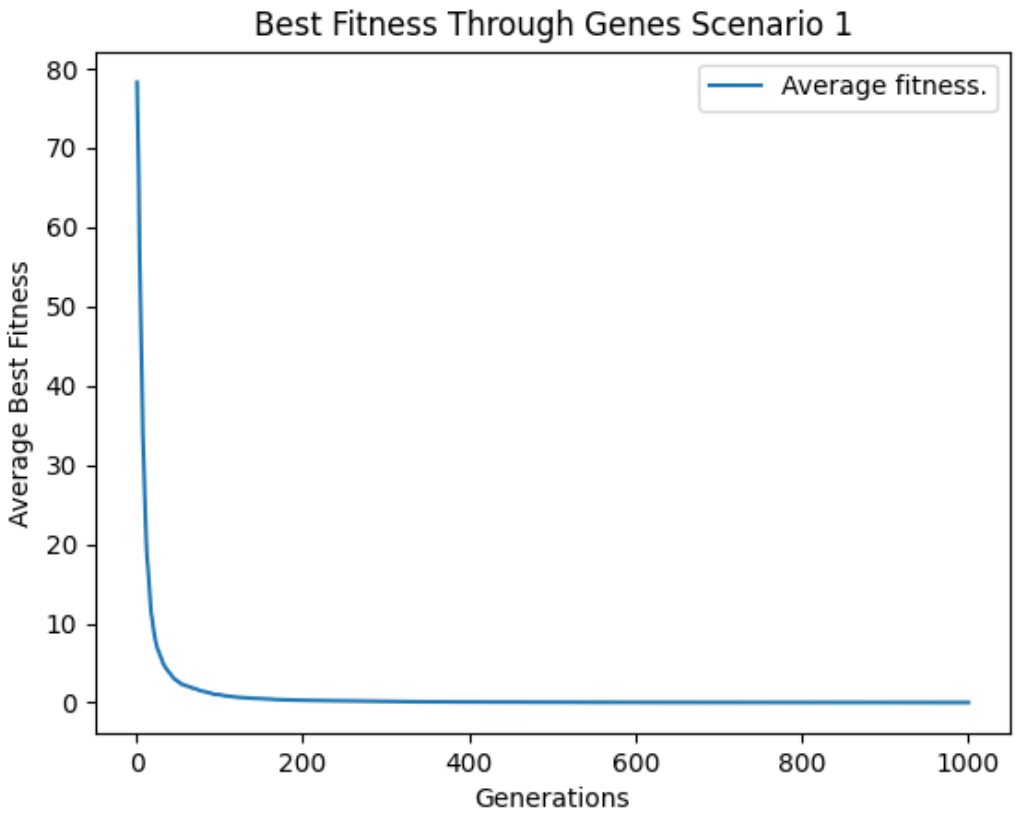
Step 2: After step one is complete for every operator, compare different operators from the same type(e.g. compare different Mutation operators in GA – Mutator vs GuassionMutator) and see which one is better.(Doesn't apply to PSO)

Step 3: Based on the values from Step 1 and 2 it is time to have some experiments with different combinations of operators and see which one gives the best fitness.

Genetic Algorithm(GA)

Scenario 1

Step 0:



As you can see in the graph above this is how the best fitness changes over generations. Note that this is the average best fitness out of 30 runs for each generation.

And as we can see graph for a million runs goes very close to zero and the exact best fit is 0.0111.

So, from here we optimize!

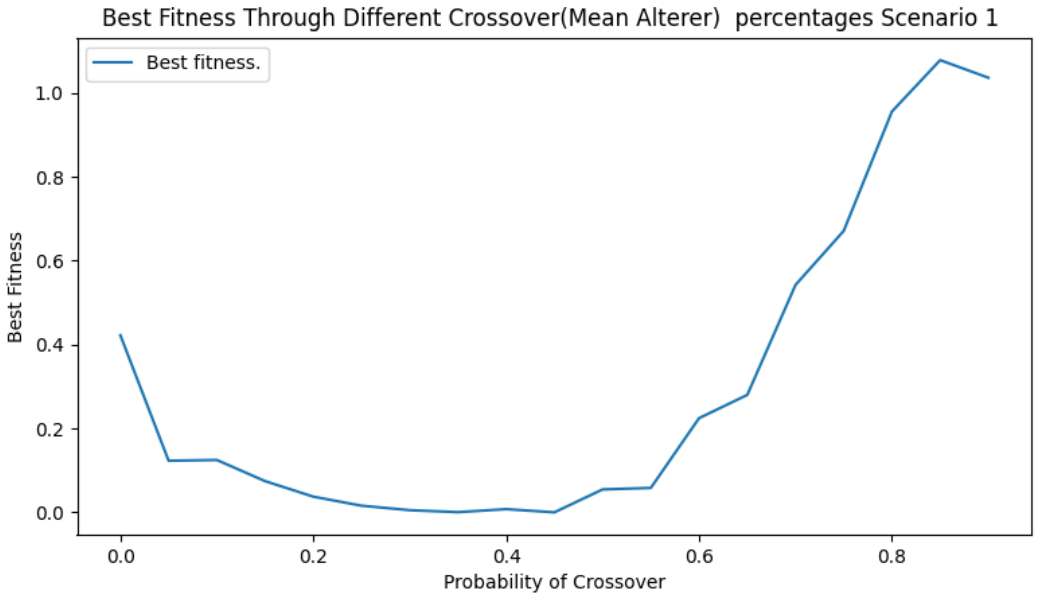
Step 1:

Note: The Achieved Best Fitness in this step is used to compare operators from the same type with each other. E.g. Gaussian Mutator vs MUTATOR(Both are mutation operators and can be compared with each other).

Crossover

There are 3 different operators for Crossover, which take the probability of crossover as a parameter(0%-100%):

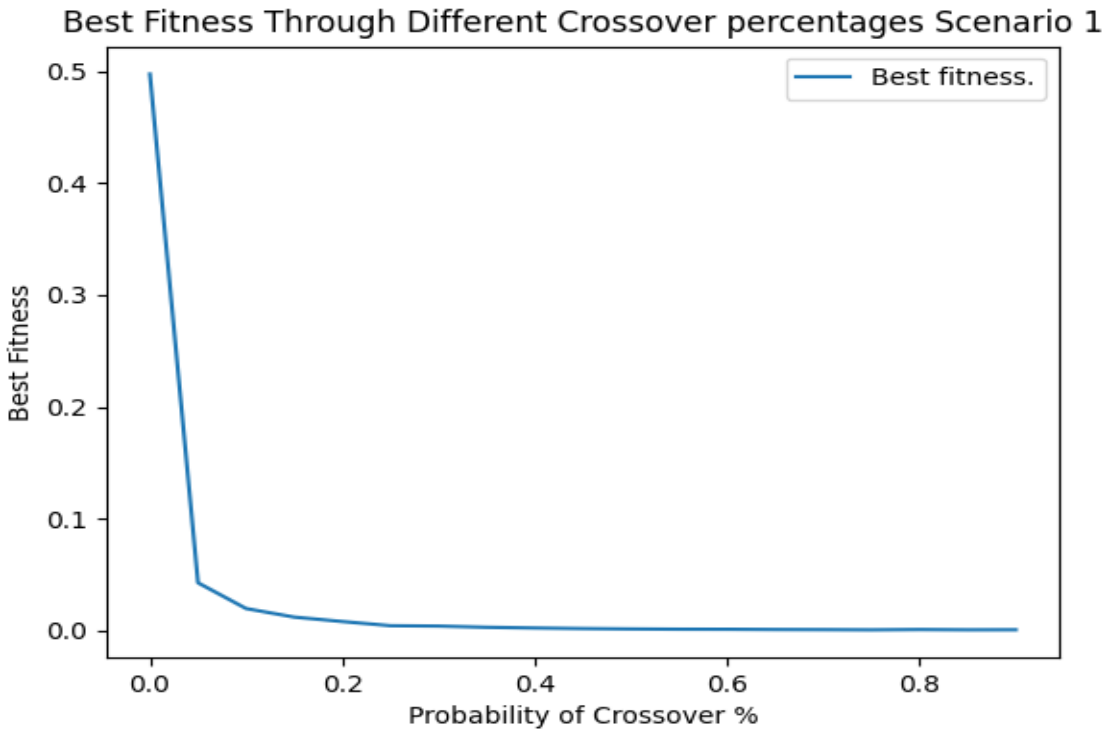
1.MeanAlterer – blends the values of the parent's genes.



On the Graph above you can see how best fitness changes for different percentages of crossover probability. We see that its lowest values are achieved between 20% and 50% .

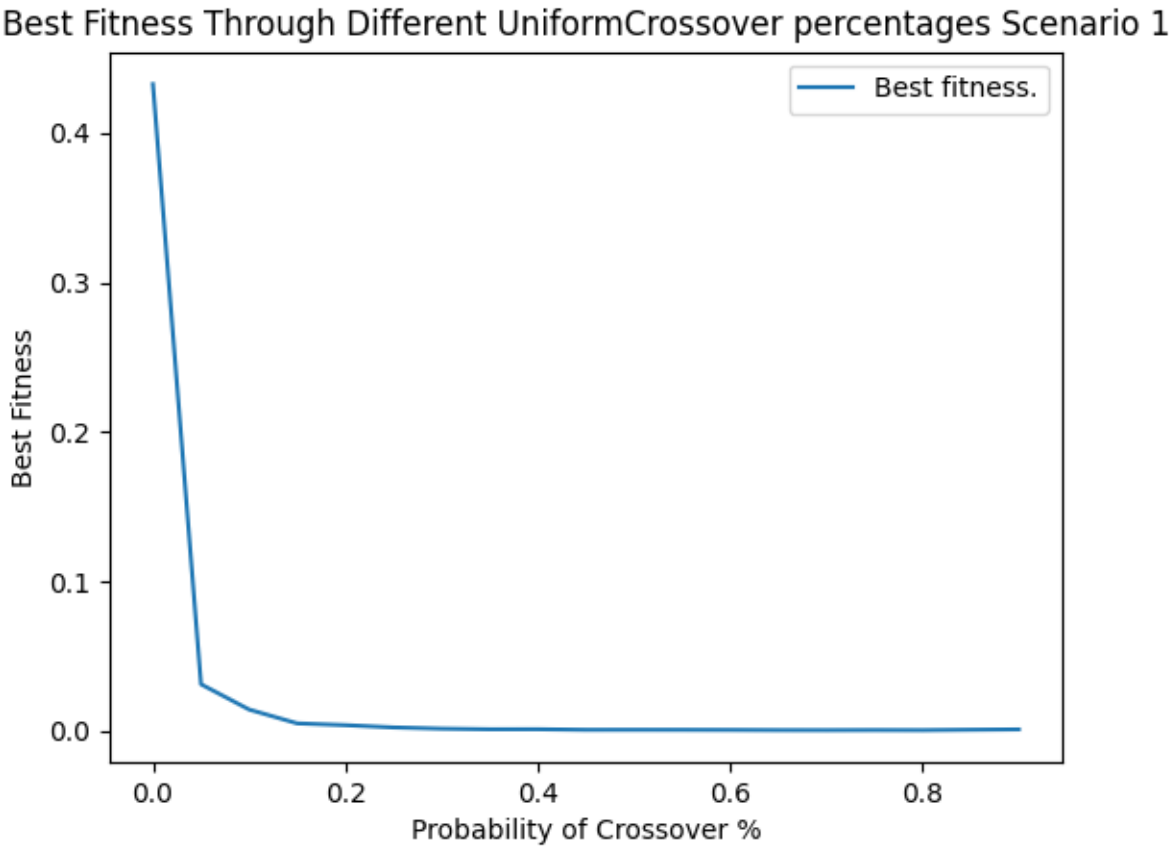
The best fitness drops down to 0.00004 at 45%.(Values are rounded)

2.Single Point Crossover - One or two children are created by taking two parent strings and cutting them at some randomly chosen site.

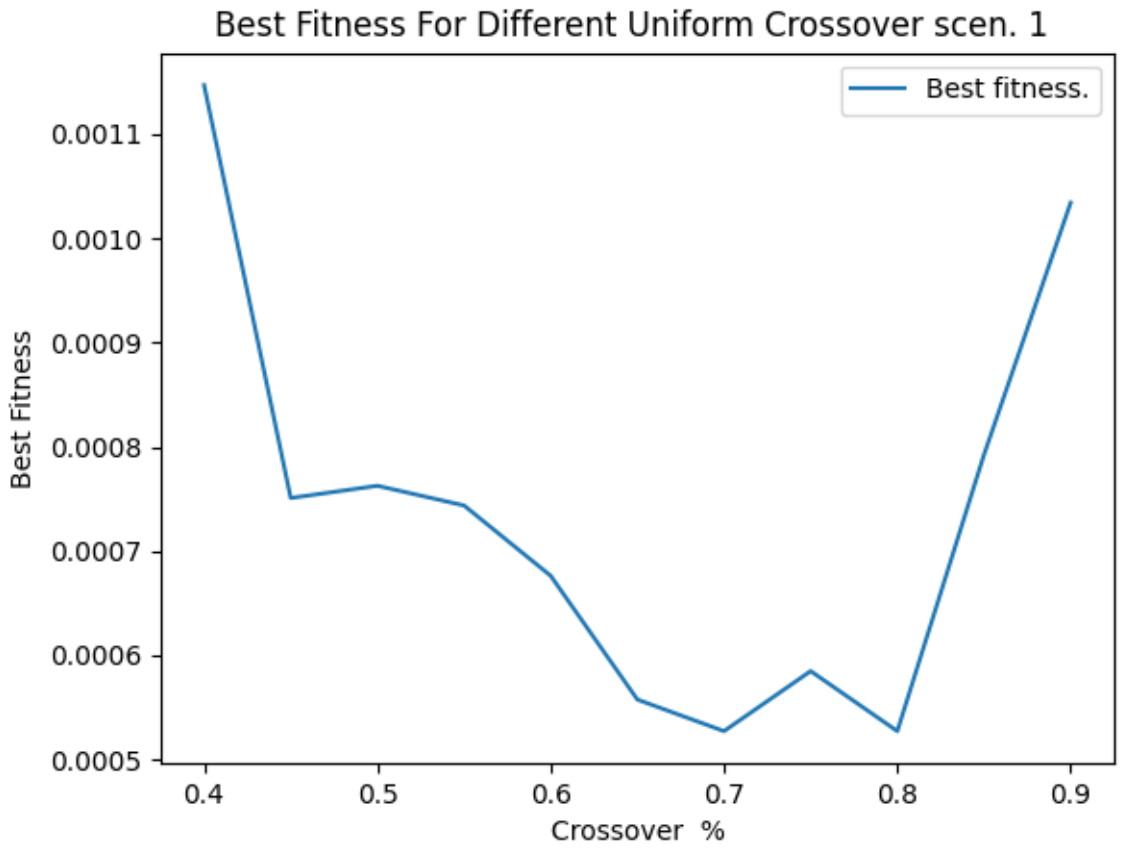


And for the best fit we achieve around 90-100% - 0.00082(Values are rounded)

3.Uniform Crossover - swaps single genes between two chromosomes



We can see that Uniform crossover acts similarly to Single Point Crossover in terms of best fitness for different percentages. The best finesses are achieved between 40% and 90% the difference it just too small to see in this graph.

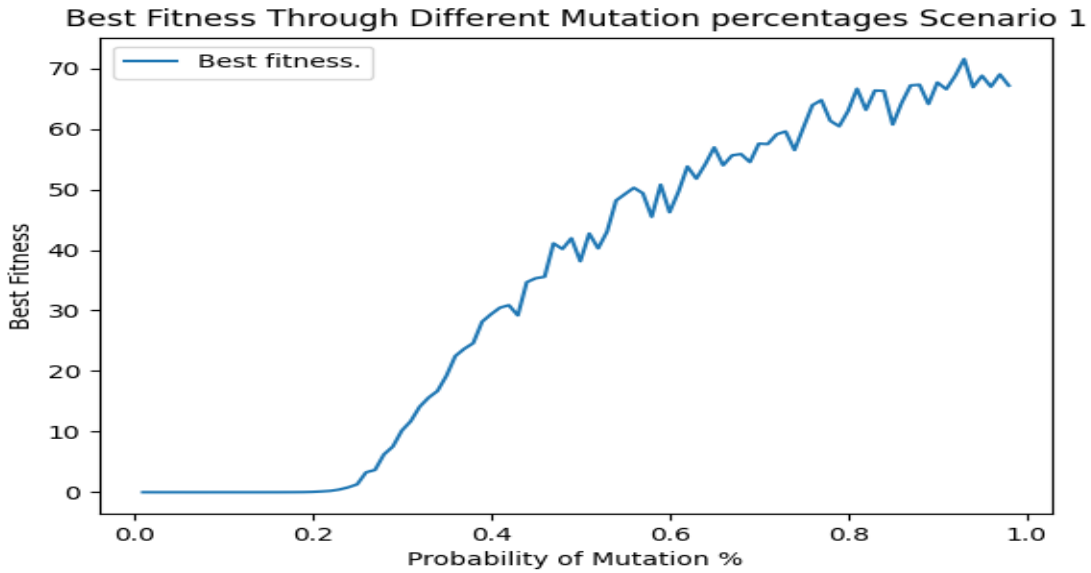


Here you can see that the lowest value is achieved at 80% and its exact best fitness is 0.00053

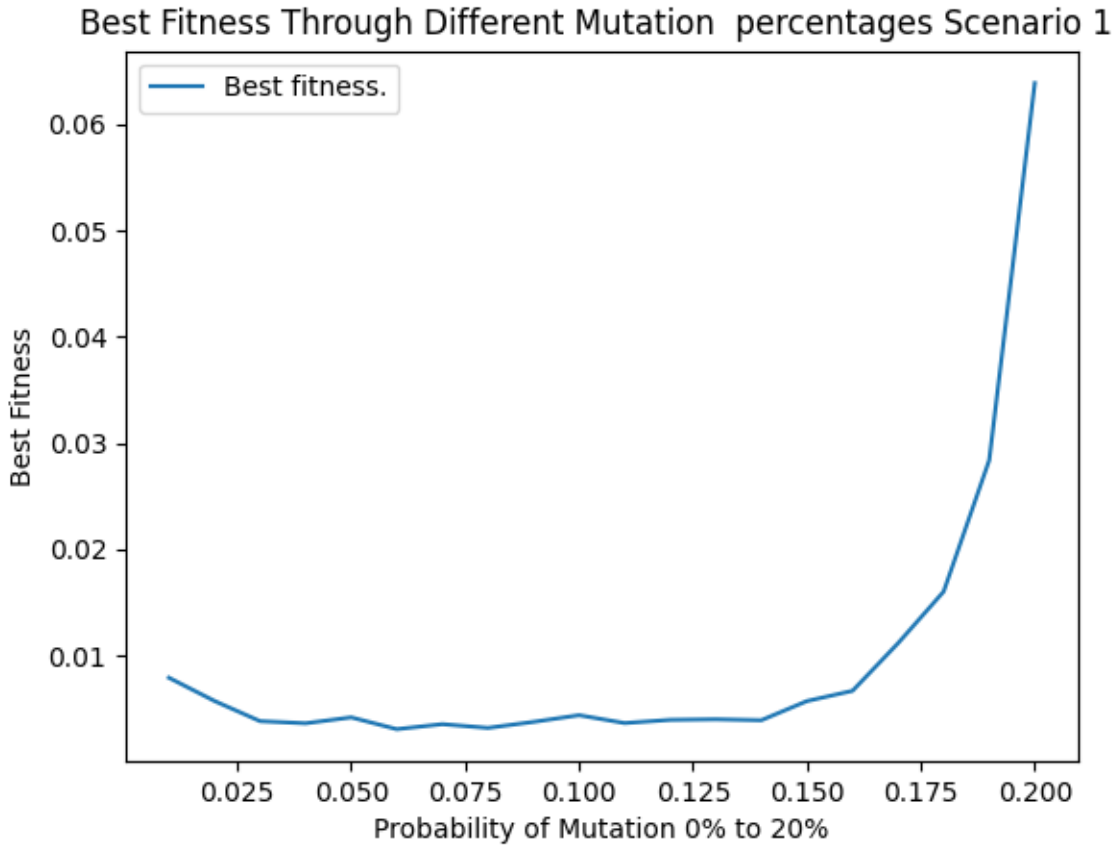
Mutation

There are two different operators for mutation – MUTATOR and GaussianMutator.

First let's look at the results from **Gaussian Mutation.**



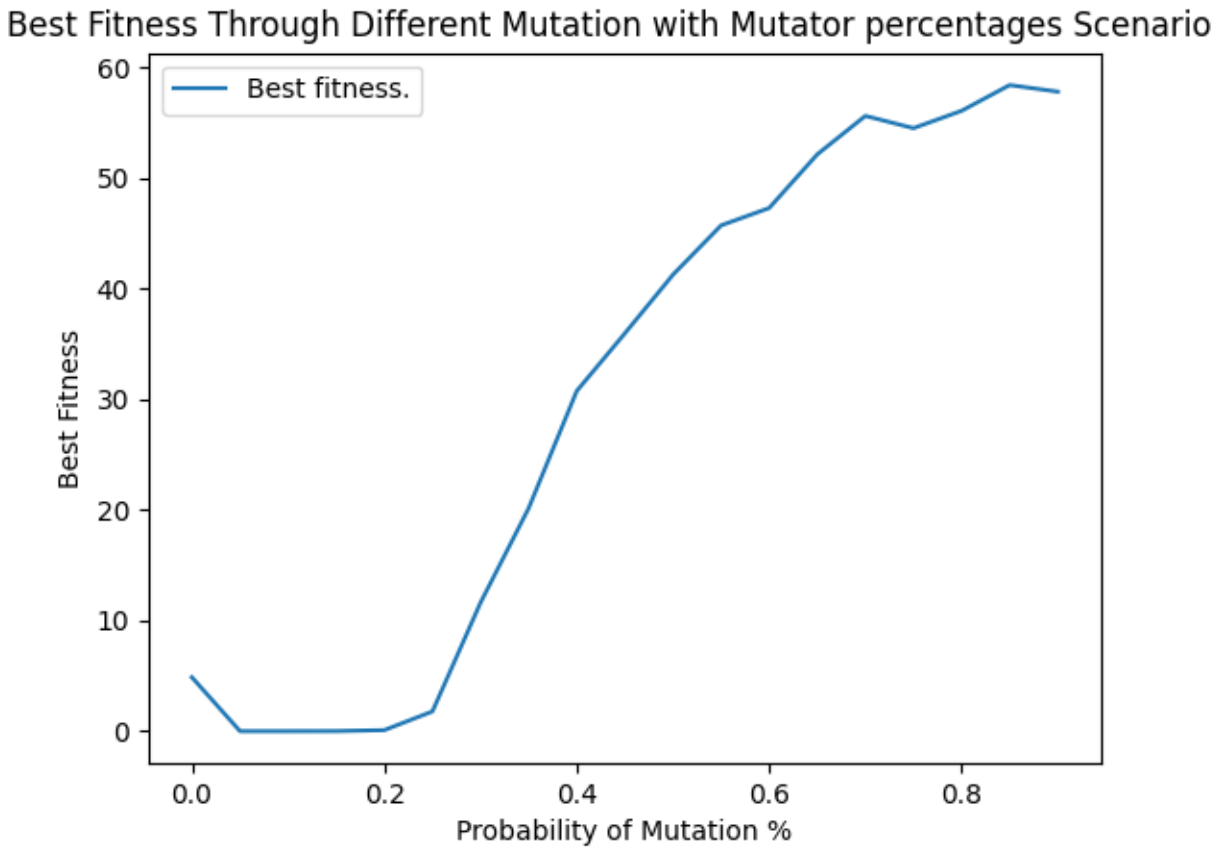
It looks like between 0% and 20% of mutation probability the Best Fitness achieved are close to zero and after that the individuals are mutated so much that they become random and harm the best fitness instead of helping.



If we have a closer look, we can see that the best values are achieved around 5% to 7.5% mutation.

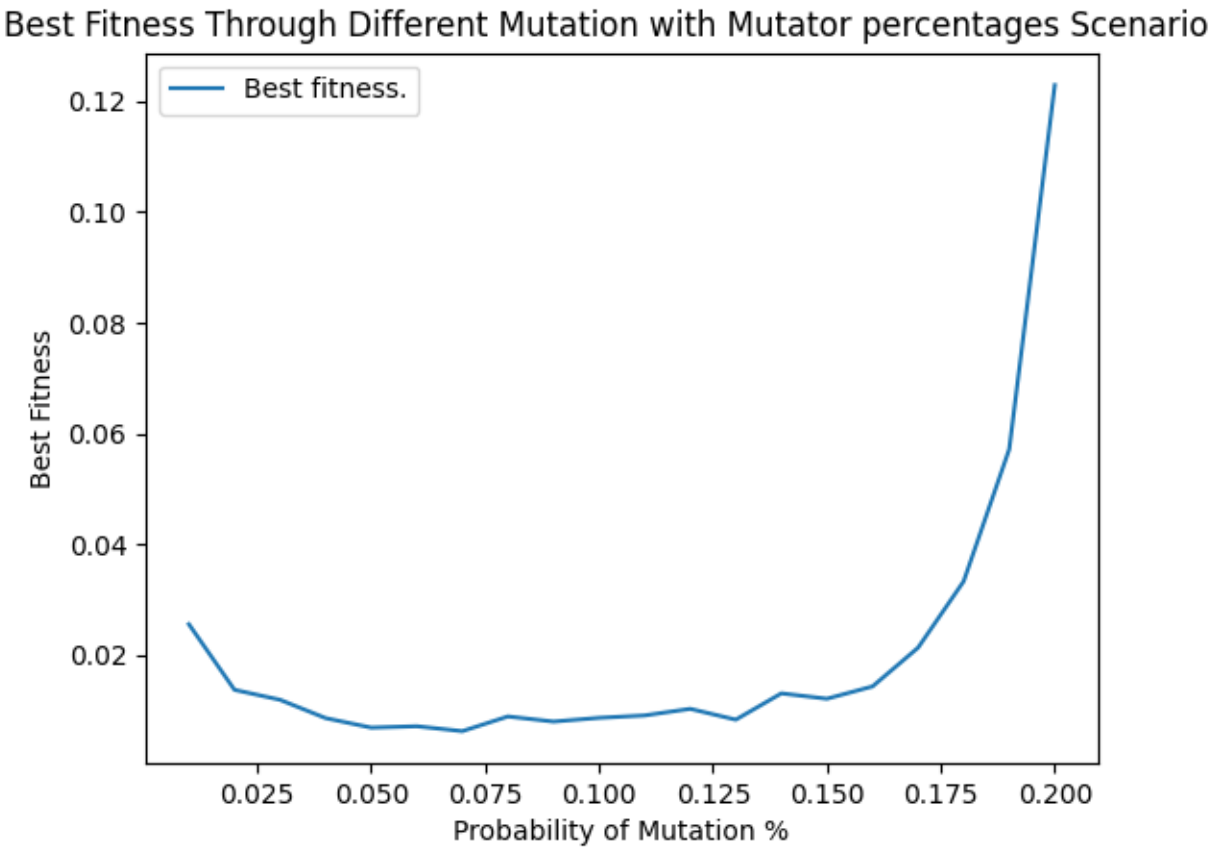
And the best achieved fitness is 0.0031 for 6% mutation probability.

Now let's examine the results from the **MUTATOR** operator



As in Gaussian Mutation the best results are achieved between 0 and 20%.

And if we have a closer look:

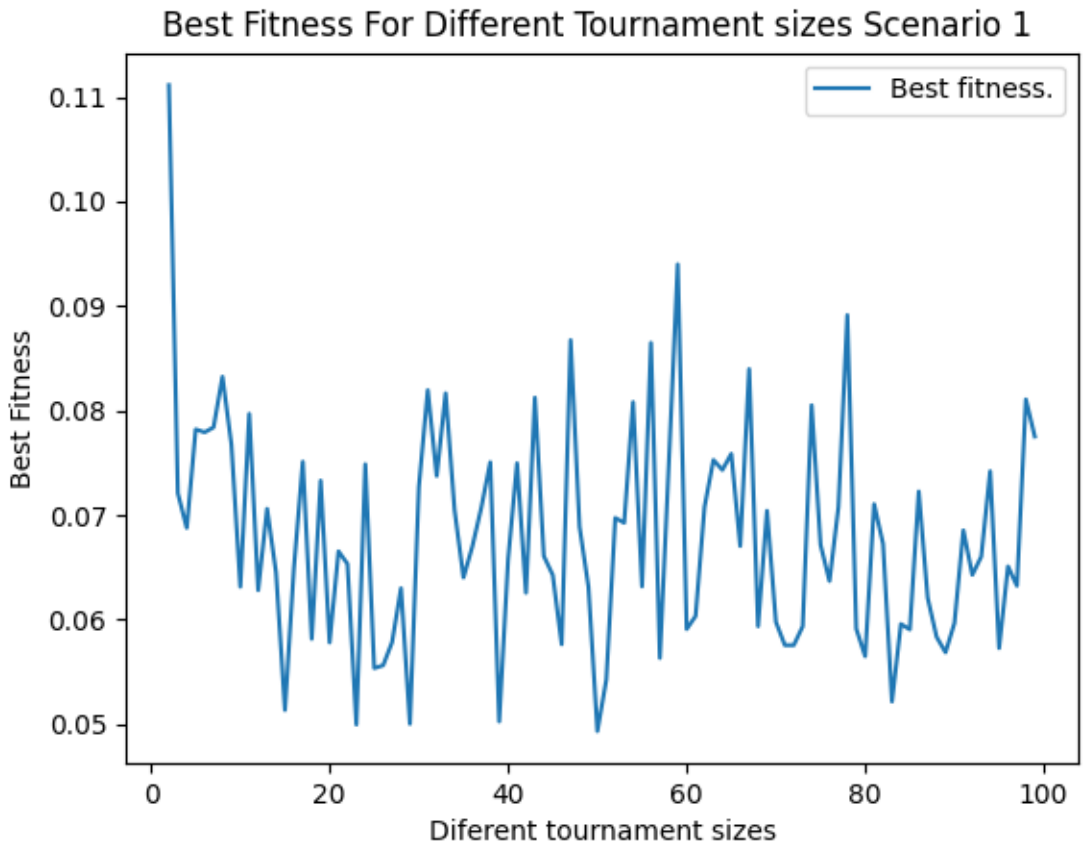


The best values are again achieved between 5% and 7.5%.

Best achieved fitness is at 7% with 0.00624

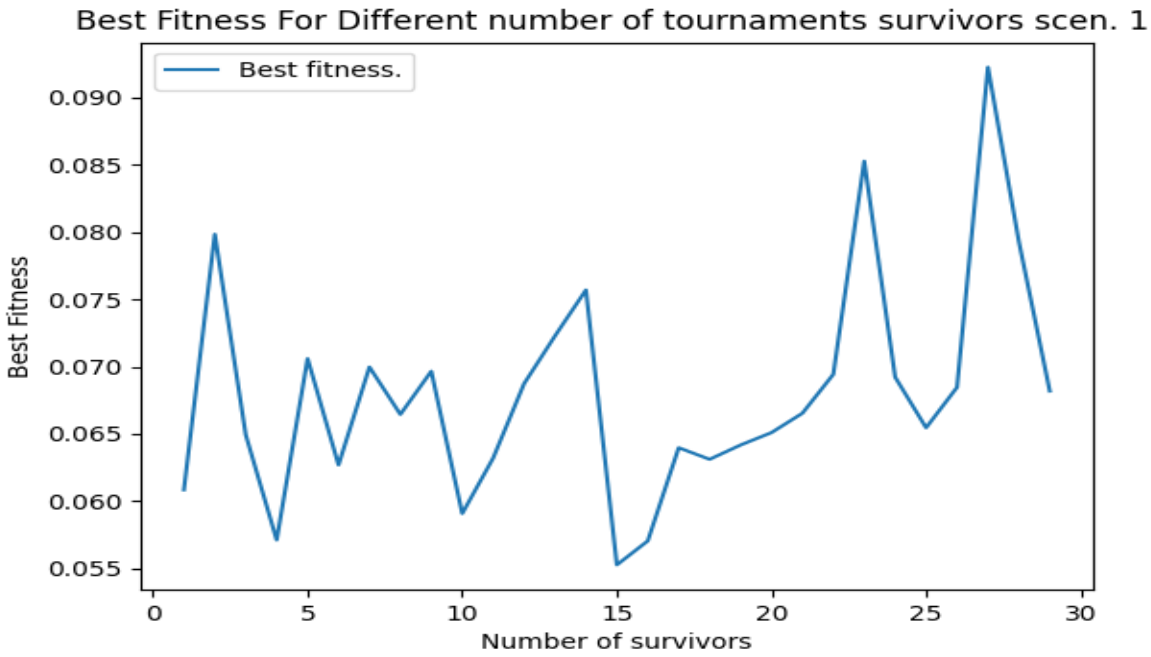
Selection

Here I test what tournament Size would be optimal for Best fitness.



And as we can see with different tournament sizes values are pretty random but best values seemed to be achieved through tournaments with sizes 25 to 30.

Also, I tested the most optimal number of survivors of the tournament:



Fitness value is the best at size 15 which is exactly 50% of the tournament to survive.

Step 2:

From the information above we can conclude that:

The best operator for **Crossover** is the MeanAlterer with 45% crossover probability

For Mutation, the two operators are really close in performance, but Gaussian Mutator showed better results with 5-7.5% mutation probability.

For Tournament Selection the most optimal tournament sizes seem to be from 25 to 30 but results are really inconsistent(it is best when survivors are 50% of tournament).

Step 3:

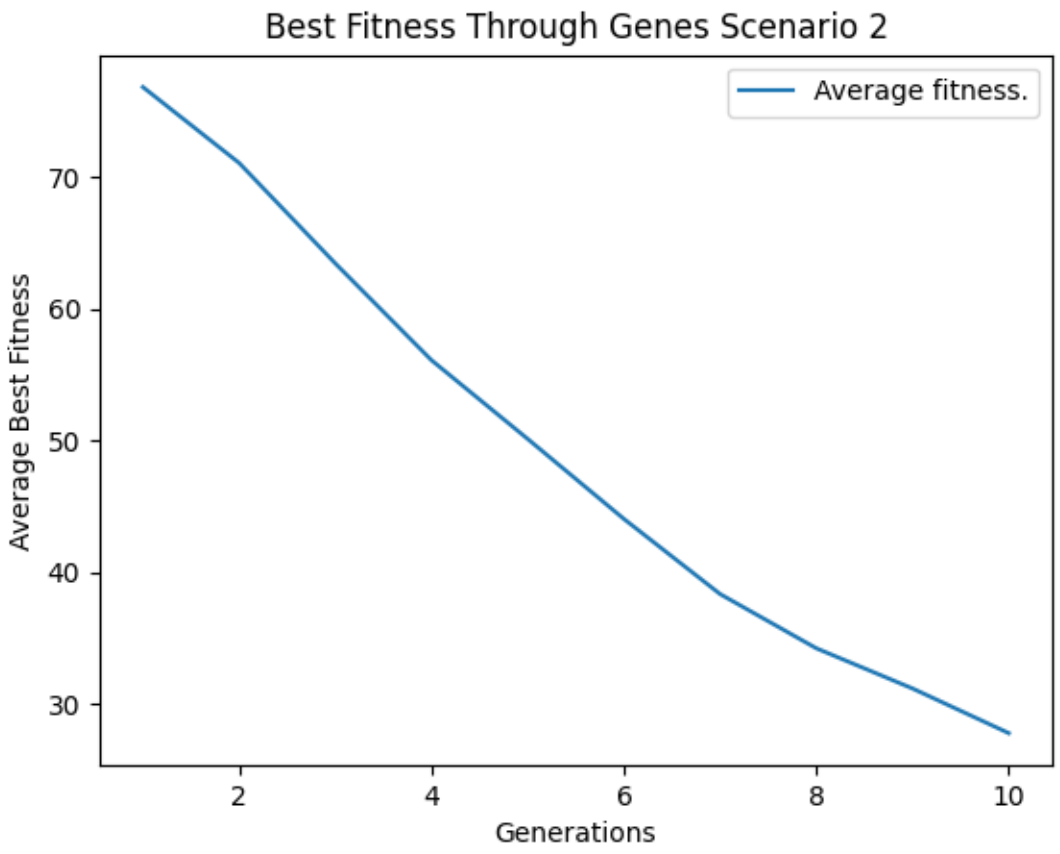
For Step 3 I have created a table with all the best value combinations and best finesses.

popSize	numSurvivors	TournamentSize	probMutation	ProbCrossover	numIters	Average Best Fitness
1000	1	2	7%	45%	1000	0.0005
1000	1	2	6%	45%	1000	0.00098
1000	1	2	5%	45%	1000	0.0004563
100	1	2	5%	45%	10000	0.00009

So, the best fitness I have achieved for scenario 1 is 0.00009 and is for the above values.

Scenario 2:

Step 0:



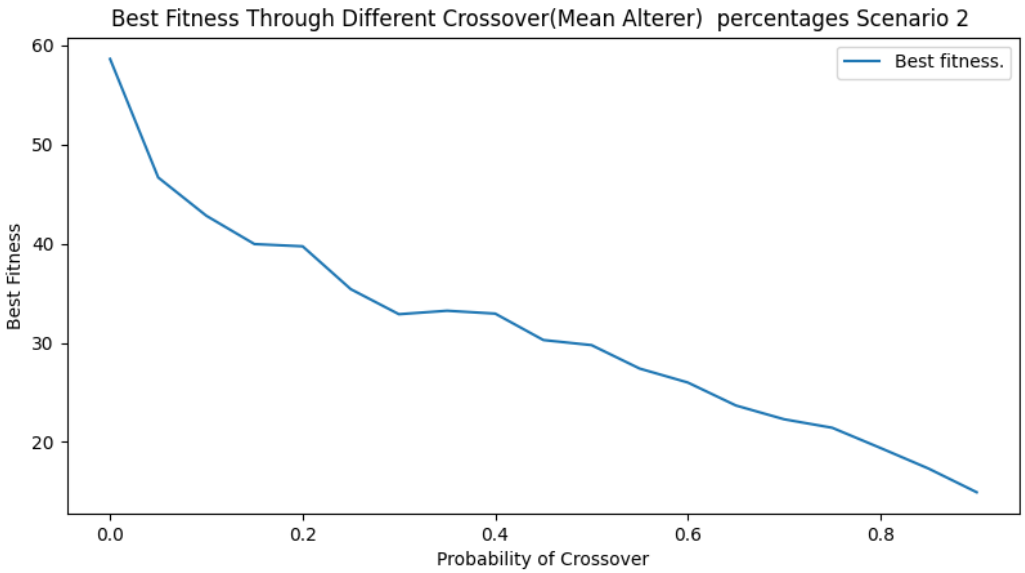
From the Graph we see that the best fit goes as low as 27 fitness.

This is the starting point of optimization.

Step 1:

Crossover:

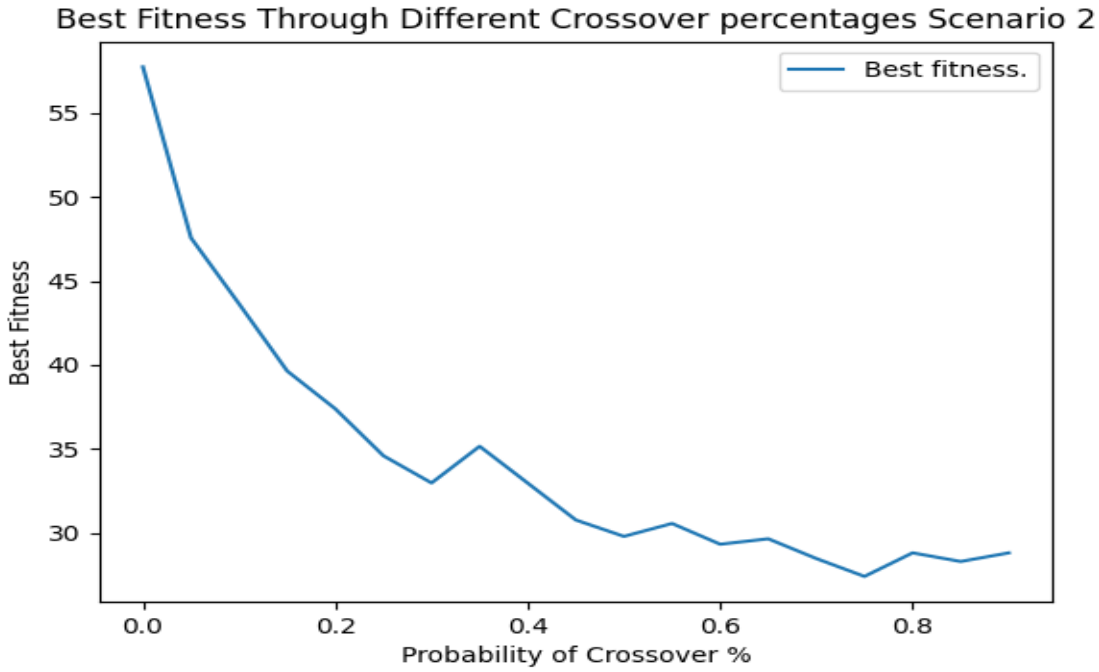
MeanAlterer
Crossover:



We can see that lower generations make MeanAlterer behave differently than in Scenario 1.

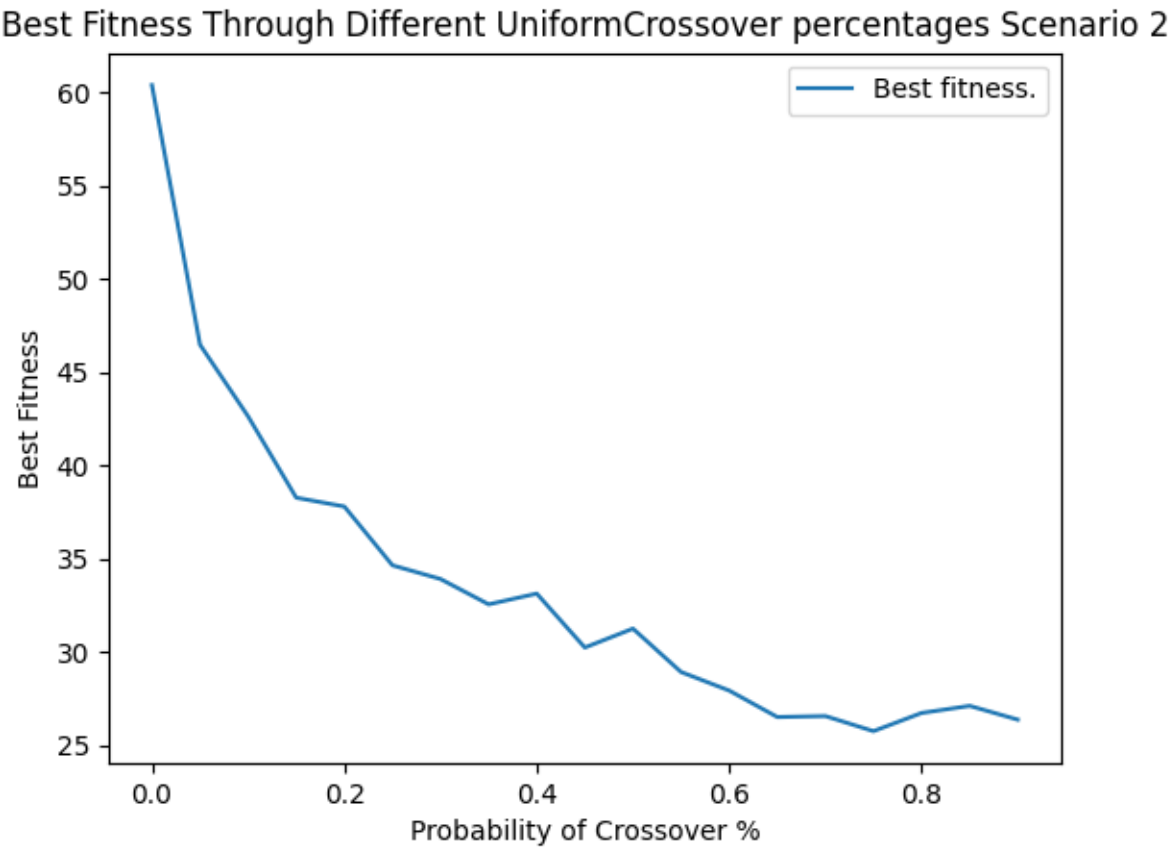
The best Fitness here drops below 15 at 90 %.

Single Point Crossover:



Here the lowest best fitness value is achieved between 65% to 80% with best fitness of 26.54

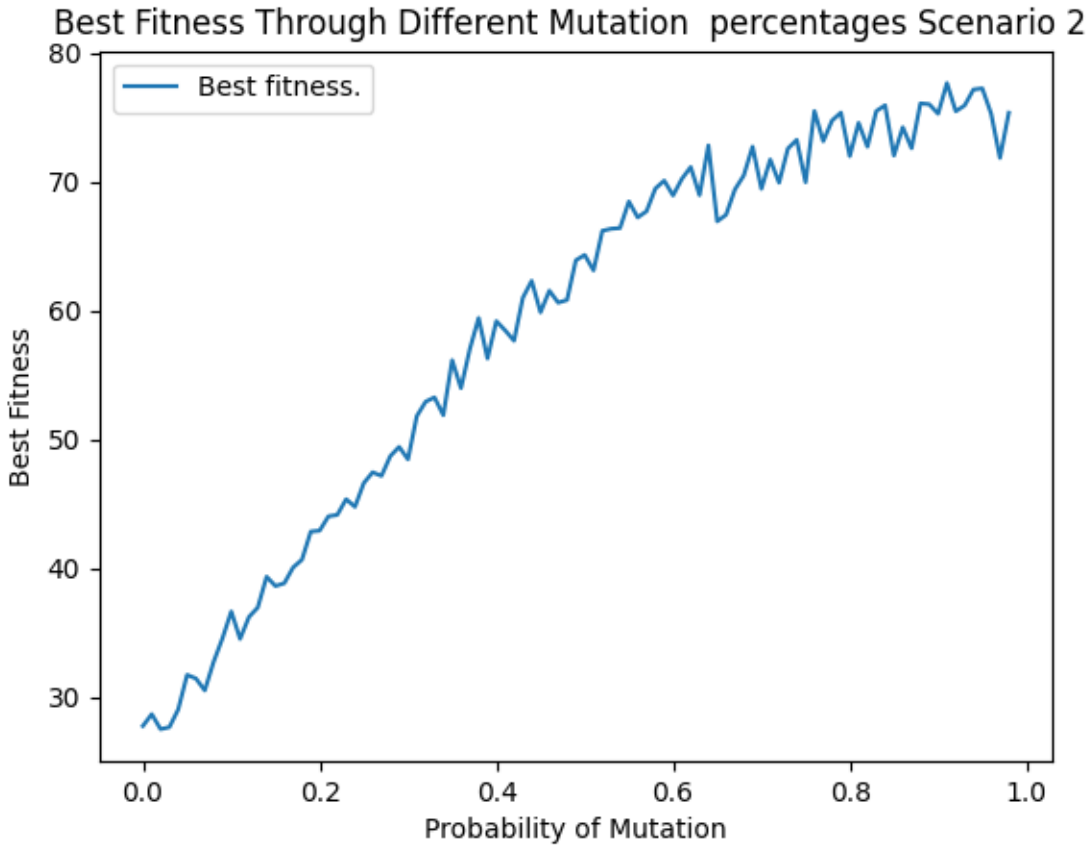
Uniform Crossover:



Here we have similar results to Single Point Crossover with the best values being between 65% to 80% with lowest best fitness value being 25.76 at 75%.

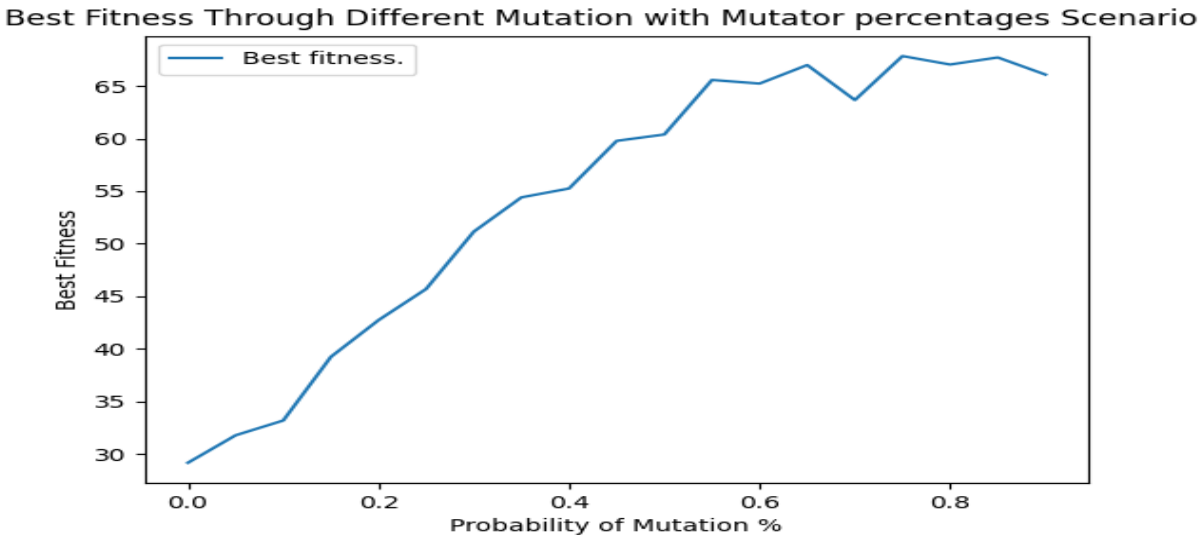
Mutation:

Gaussian Mutation:



We see that mutation acts similarly as in scenario 1 – If we increase it too much it starts to harm the best fitness instead of helping.

We can see that for the MUTATOR operator also:

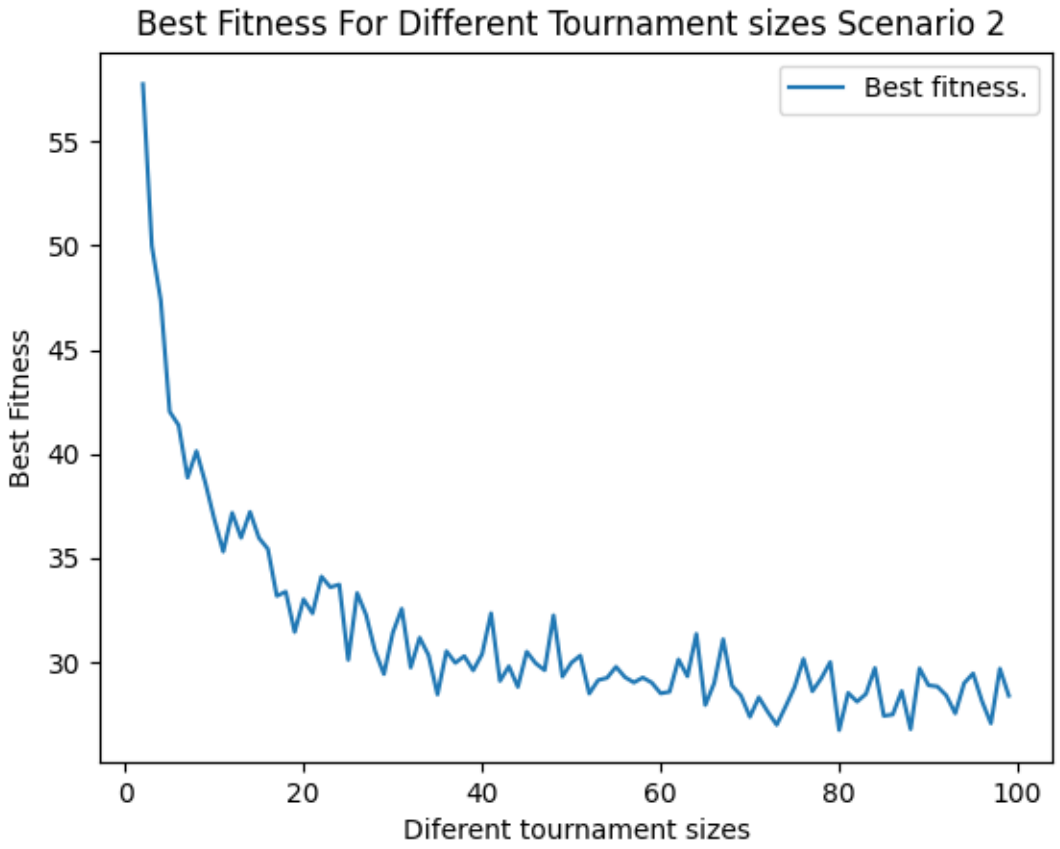


As a conclusion we can say that with smaller iterations(generations) we have to use smaller percentage of mutation.

The best fitness doesn't improve with mutation for the current iterations as the lowest fit for Gaussian mutation is 27 and for MUTATOR is 29.

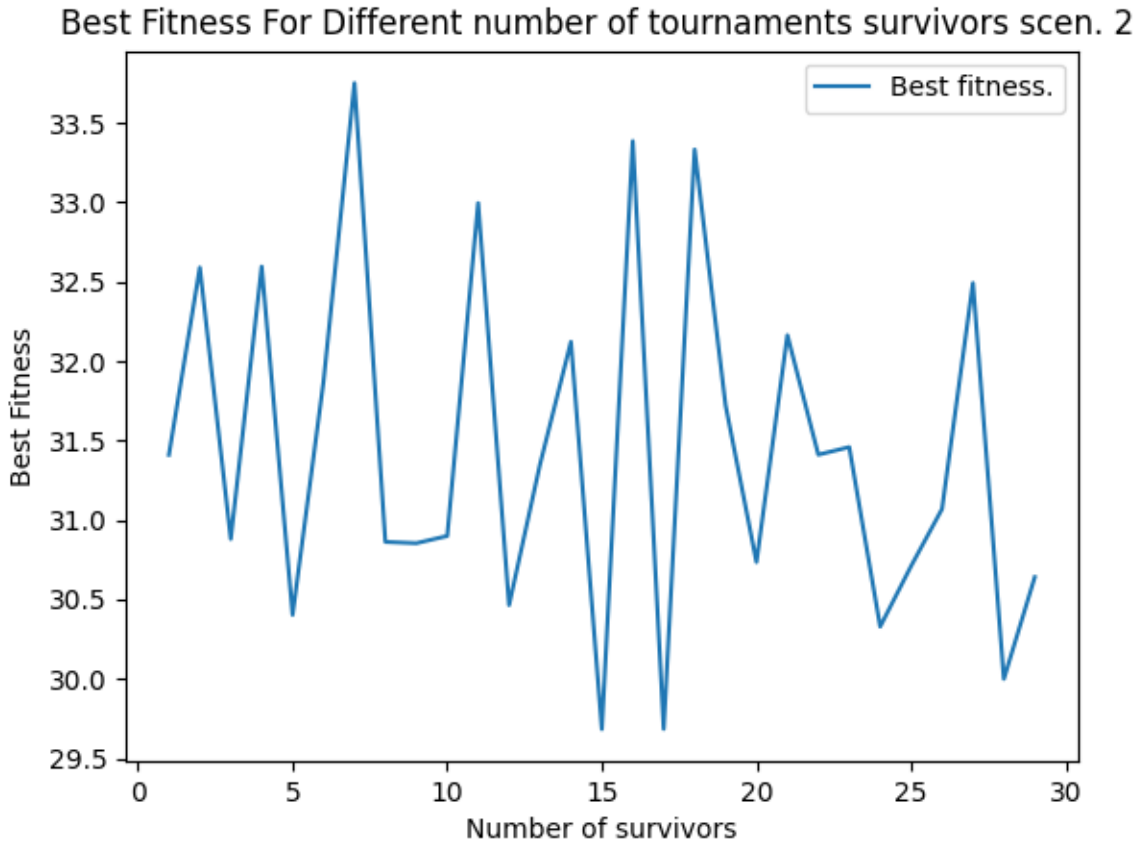
Selection

In contrast to scenario 1 different tournament size seems to help in getting better results!



With the best tournament size 80.

Also, from the graph for the number of survivors of the tournament:



We can say that tournament selection achieves its best results when half of the participants in the tournament survive.

Step 2:

For crossover best results gave again the MeanAlterer at 90%-100% probCrossover

For mutation, the best results were at 0%-0.5% with GuasianMutator.

Increasing the size of the tournament in selection seemed to help in scenario 2 with best at 80.

Step 3:

First, I try with best values from step two and then I 'mix and match' until I find the best result.

popSize	numSurvivors	tournamentSize	probMutation	probCrossover	numIters	Average Best Fitness
1000	1	80	5%	90%	10	15.27
1000	40	80	5%	90%	10	14.34
1000	1	2	5%	90%	10	6.00
1000	1	2	5%	95%	10	5.75
1000	1	2	4%	95%	10	4.73

I also tried different variations of iterations/popSize ,but they didn't seem to help the best fitness, so I didn't include them.

Tournament size didn't seem to matter as I thought In step 1.

In the end I have got the best fitness down to 4.73 for scenario 2.

Particle Swarm optimization(PSO)

Step 0:

Step zero I test with the default values of the config files.

```
numParticles=1000
numIters=10
neighWeight=1.0
inertiaWeight=1.0
personalWeight=1.0
globalWeight=1.0
maxMinVelocity=0.0001
```

For Scenario 2 the average best fitness is `18.39339769303924`

```
numParticles=1000
numIters=1000
neighWeight=1.0
inertiaWeight=1.0
personalWeight=1.0
globalWeight=1.0
maxMinVelocity=0.0001
```

For Scenario 1 the average best fitness is `10.33292321149021`

Step 1:

The first Variable tested is maxMinVelocity.
The values for 1,000,000 runs draw this graph
I tested minMaxVelocity in steps of orders of magnitude.
We can see from the graph that increasing the maxMinVelocity variable doesn't make best fit better.

```
10.0,14.938385552320586
1.0,14.938385552320586
0.1,10.83106700582723
0.01,9.95107442368347
0.001,10.57974250348988
0.0001,11.110365473760503
0.00001,9.816602554203905
0.000001,11.346602554203905
0.0000001,10.816602554203905
```

And decresing it doesn't help eather.
The same thing can be seen in the example for 10,000 runs.

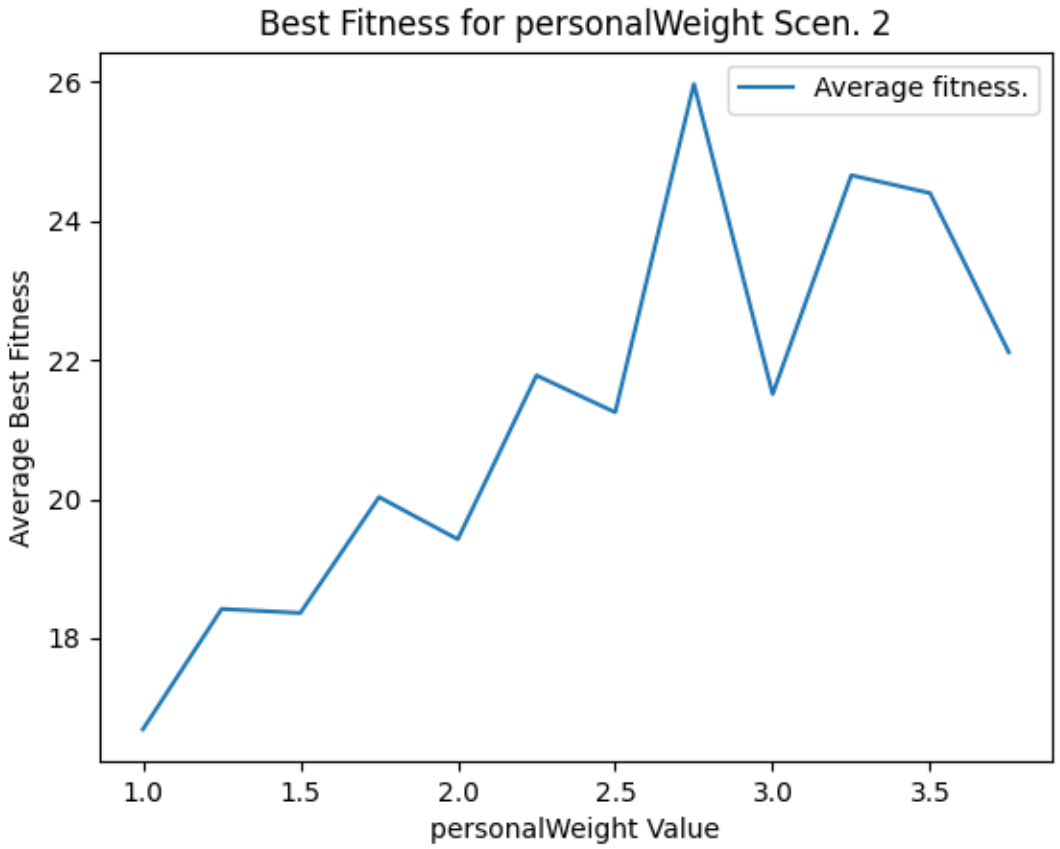
(I couldn't draw a graph here as the software I wrote for drawing was drowing nosense for such ranges of values.)

Next graph is for 10,000 runs of the algorithm.

```
10.0,73.8338870748906
1.0,34.8981877061452
0.1,18.403534202994198
0.01,19.69558258433536
0.001,17.525729942140128
0.0001,16.650010248984934
0.00001,20.06881026338987
0.000001,20.43011572460164
0.0000001,17.834887080256372
```

Next I tested all 4 of the weights in ranges 1.0 – 4.0 with step 0.25.

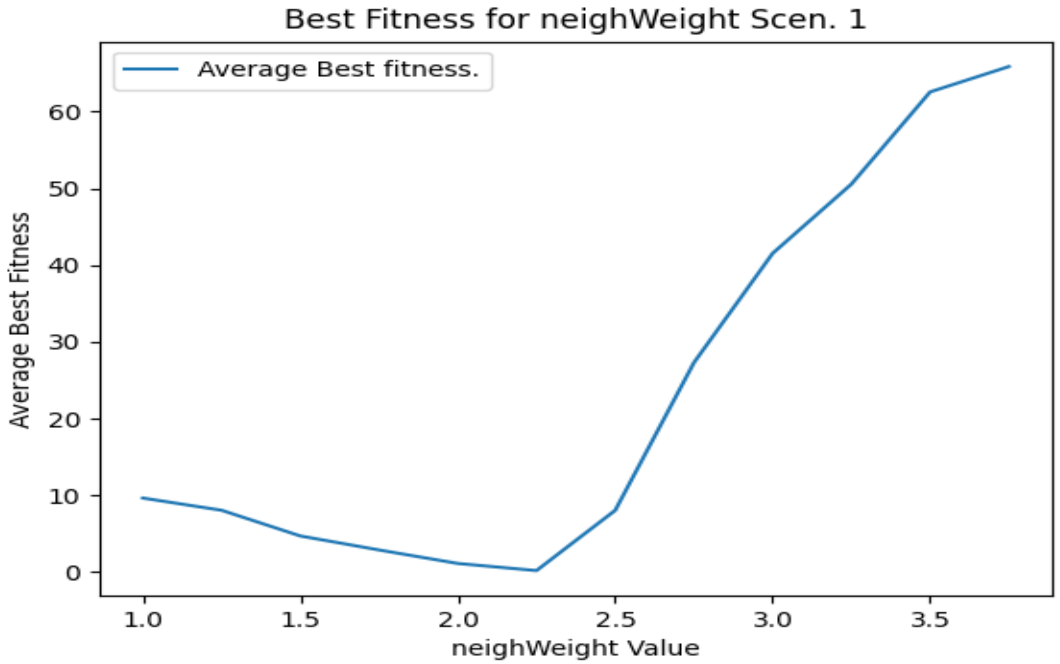
First we look at personalWeight for scenario 1 and2

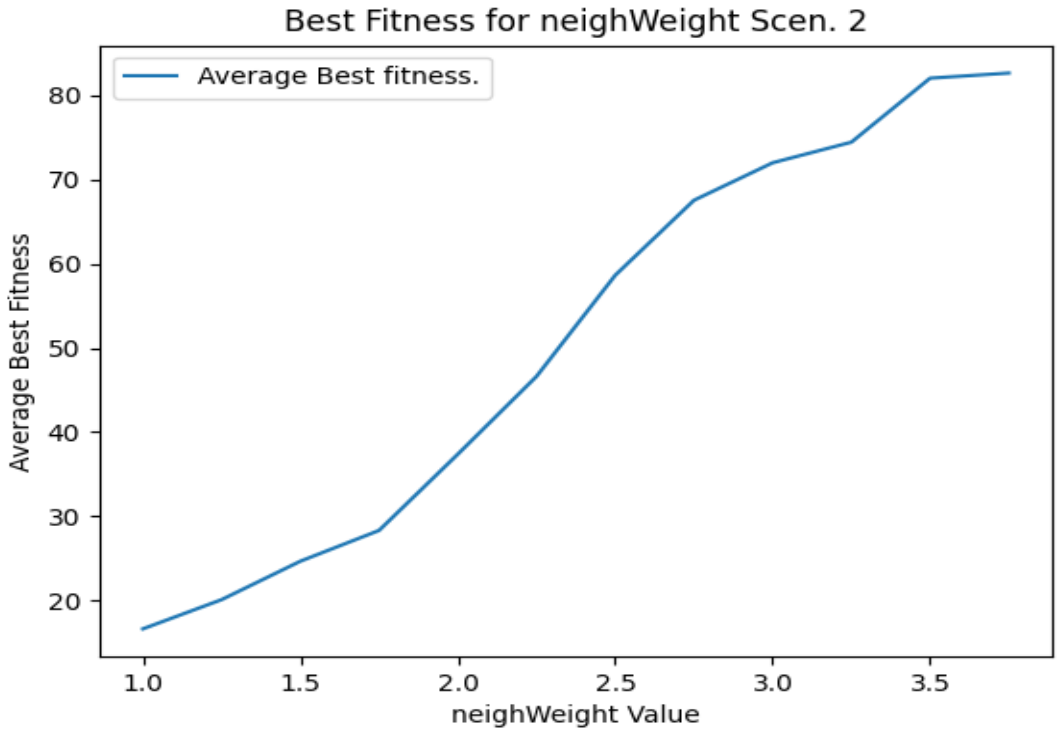


:

From the graphs we can clearly see that in scenario 1 we get the best fitnes for personal weight 2.5 and in scenario 2 increasing the personal weight value actually hurts Best fit, so the best value is at 1.

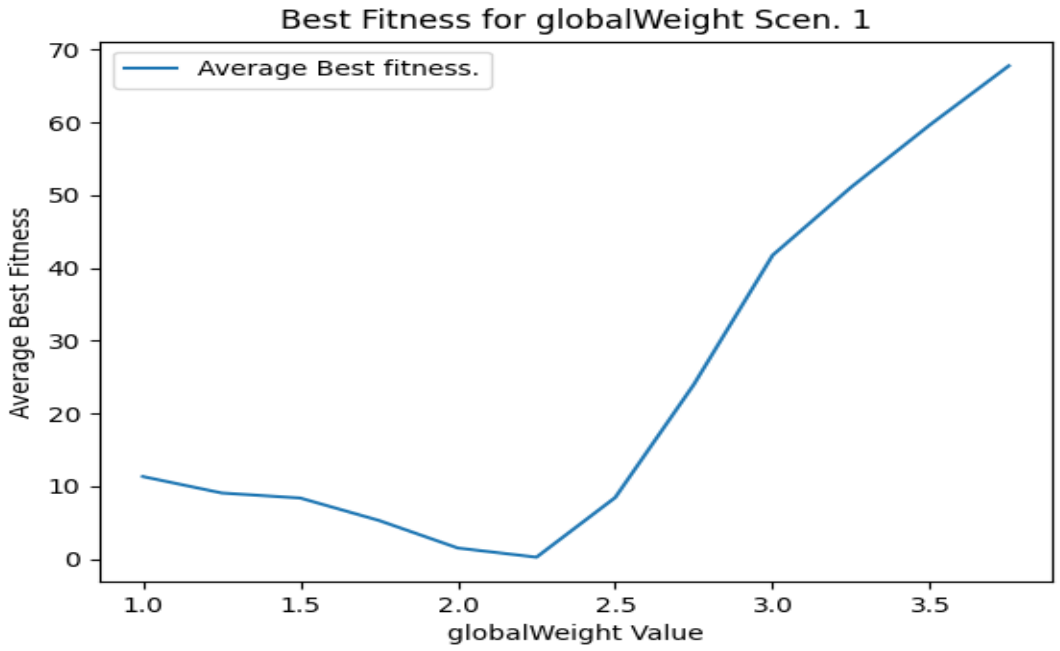
Next we look at neighWeight for scenario 1 and 2

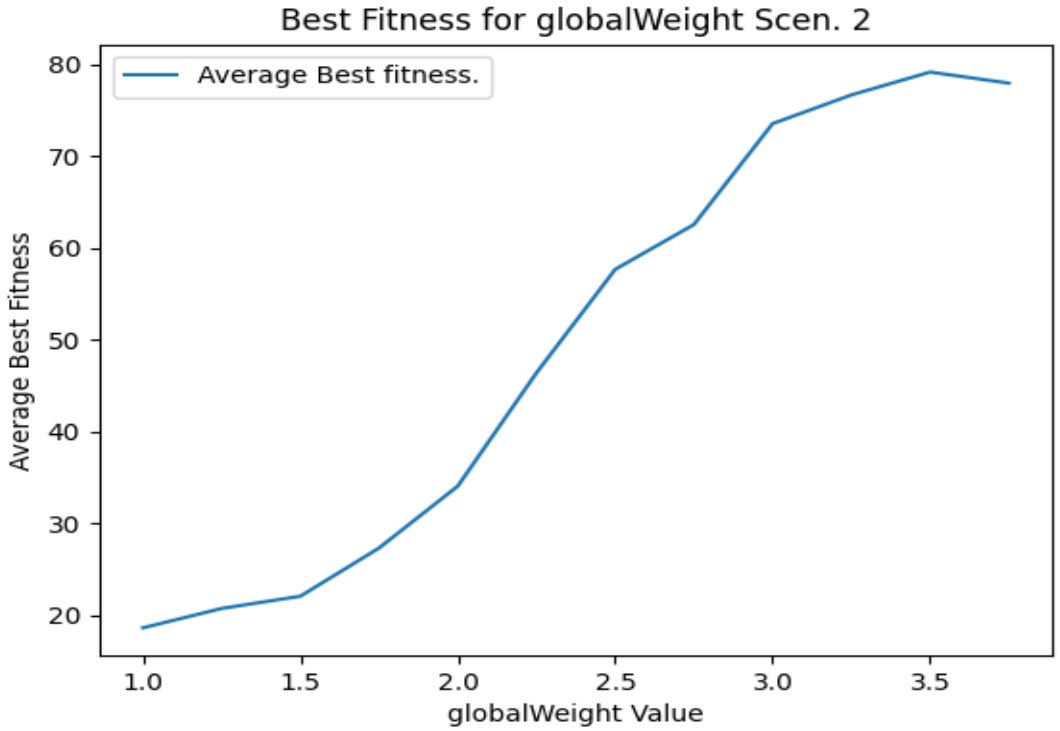




From the graphs we can see that in scenario 1 we get the best fitnes for neighWeight 2.25 and in scenario 2 increasing the personal weight value hurts Best fit, so the best value is at 1.

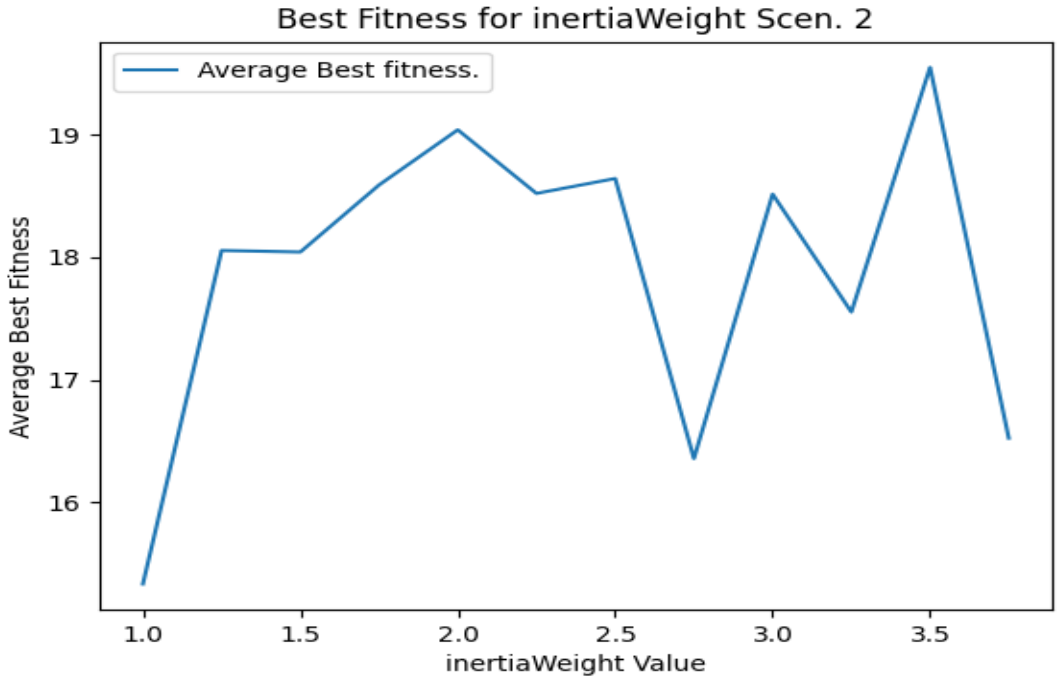
Here I test global Weight for scenario 1 and 2 and we get these graphs.





We can see almost the same pattern as in neigh Weight. Here as in neigh Weight the best values for fitness are for 2.25 global weight(scenario 1) and for 1.0 global weight(scenario 2)

Last but not least here are the graphs for inertiaWeight for scenario 1 and 2.



As we can see from the graphs the best average fitness is for 3.25 inertiaWeight in scenario 1 and in scenario 2 the best results are achieved at the beginning for 1.0 initaWeight.

Now that we have done some parameter sweeps, we can continue onto testing different combinations.

Step 2:

We don't have step 2 here, because we don't have operators only parameters.

Step 3:

Here we test different combinations of parameters and see which one gives the best fitness.

Scenario 1:

neighWeight	inertia Weight	personal Weight	Global Weight	maxMin Velocity	numIters	numParticles	Best Fitness achieved
2.25	3.25	2.5	2.25	0.0001	1000	1000	83.22
2.25	1	1	1	0.0001	1000	1000	0.27204644471683453
2.25	1.25	1	1	0.0001	1000	1000	0.07570856303298187
2.25	1.25	1	1	0.0001	10000	100	0.00000057340911426
2.25	1.25	1	1	0.0000001	10000	100	5.179856543691131E-13

In the first example we see that just combining the best values of the parameters doesn't give us a good result. This leads us to the conclusion that in order to optimize we must use the value of the weight that gives the best results and adjust the other weights, so we get better results.

So next we set neighWeight to 2.25 and everything else to 1. Here we get a 0.3 fit.

If we increase inertia Weight to 1.25, we get our best fit down to 0.1.

Every time we increase global weight or personal weight best fit gets worst.

So, I this point I started to play with the number of iterations and the number of particles.

And I have got amazing results! I got the best fitness down to 5.73409114260598E-7.

Even better best fitness I achieved is 5.179856543691131E-13 and it was with the same value just the min maxVelocity was 0.0000001. With the same values I have achieved also a 0.0, but I don't want to put it in the table, because its inconsistent. So those are the best results I achieved.

```
Best fitness: 0.0
Best position:  [-7.842437856028
Number of evaluations: 1000000

globalWeight =1.0
best average fit is 0.0
```

Scenario 2:

neighWeight	inertia Weight	personal Weight	Global Weight	maxMin Velocity	numIters	numParticles	Best Fitness achieved
1	1	1	1	0.0001	10	1000	19.634435821776165
1	1	1	1	0.000000 00000001	10	1000	17.30788192592132
1	1	1	1	0.000000 00000001	100	100	14.739419553516113
1.9	1	1	1	0.000000 00000001	100	100	9.002057247672049

From the individual sweeps for scenario 2 increasing any of the weights does not help with improving best fit. This is why here the difference will make setting the right min max velocity and iterations/particles ratio.

The best results that I managed to get were to get the best fit down to 9.

PSO and GA comparison.

After optimizing both algorithms I can say that PSO is a lot easier to optimize, because it does not have any operators it only has parameters that must be adjusted to achieve the best results. In GA on the other hand, we have a lot more that we can change, so It is easier to optimize in scenario 2,where number of particles and iterations is smaller. And as we can see from the results the best solution for scenario 2 is done using GA. However, PSO is performing a lot better with a large budget of fitness function calls, which makes it a lot better at Scenario 1.

Also, I can say that PSO is definitely takes much less time to execute compared to GA .

Another difference between the algorithms is the way of sharing information. GA's chromosomes share information with each other, but in PSO the best particle in a neighborhood or globally informs the others.