CS 420
Project 5
Ksenia Burova
April 23rd, 2017

Genetic Algorithms

Project Report

***Abstract*:** In this project, the goal was to investigate details of the simplified *genetic algorithm* (GA) with emphasis on important biological processes like variation, heredity and selection. The following set of parameters was observed in different variations: population size, number of genes in individual, mutation and crossover probabilities, and number of generations. For every set variation, the program was run several times to better illustrate results and statistics. The discussion of the effects of the various parameters and overall ability to find the best solution concludes this report.

1. Introduction and the algorithm.

The methods of adaptation like variation, heredity and selection, are motivated by biological evolution. The essential parts of adaptation procedure are survival of the fittest and genetic diversity. In computer science, the purpose of genetic algorithms is to simulate real biological evolution by mapping program and data on DNA-like structures. "The DNA-like structures exist in populations those members can mate, cross over, and mutate, thus sharing fitness-increasing traits similar to those of real populations of species found in nature" (Flake, 342). Genetic algorithms are widely used for solving optimization problems, where optimization problem is a problem of finding the best solution out of all. Fitness function used in algorithm quantifies merit of potential solutions.

The algorithm itself includes following steps:

a) Each "individual" is represented as genetic string of '0's and '1's, the length of the string represents number of genes. Population gets initialized randomly with the number of such "individuals".

b) *Fitness* of each "individual" gets calculated and evaluated. In case with binary strings, fitness is defined as a percentage of correct characters, whereas correct characters are '1's. Defining fitness is very important because method of selection and the way GA will work depends on it.

To find raw fitness, we divide number of correct bits by total number of genes. The problem is that if strings become larger, it becomes harder to distinguish strings that differ by one bit. To make

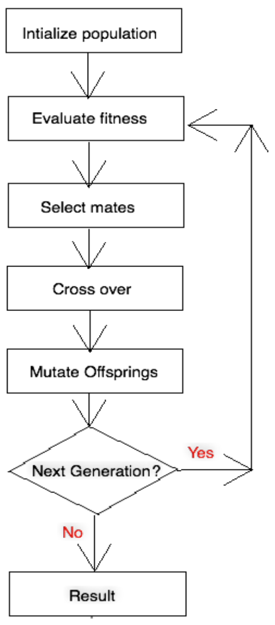


Figure 1. Genetic algorithm outline.

fitness invariant in the length of the target string we calculate scaled fitness by computing $f_{scaled} = 2^{f_{raw}}$. And then we normalize it, so selection decisions can be made regardless of f magnitude.

c) A pair of parents chosen several times to mate and cross over. At first, we could choose top 50% of the fittest "individuals", but unfit "individuals" may still have important genes, plus we don't want to exclude diversity. Instead, we use an array of sums of normalized fitness values, such that each sum for i^{th} "individual" includes a sum of all normalized fitness values for preceding individuals. Then we choose random number, and its upper bound in that array will correspond to one of the parents. We choose 2 distinct parents this way.

   After choosing parents, depending on crossover probability value, we decide to cross them over or not when mating. If we do, we flip parents' genes after crossover point as shown in Figure 2. We implement one-point crossover, and crossover point is also chosen at random.

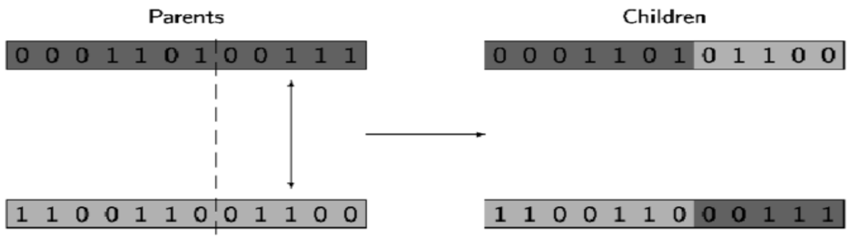


Figure 2. Crossover operation. Source: researchgate.net

a) Next, we mutate resulted offspring depending on mutation probability. Mutation allows to apply new genetic materials to future generations. Nevertheless, it more often has negative effects rather than positive, so we don't want to mutate every child completely, instead we apply mutation operation bit by bit checking probability value first.
b) Finally, we replace parents' population with offspring's' population.
c) We repeat process or not depending on number of generations.

2. Experiment Set Up.

The following parameters are used in this experiment:

a) Number of genes in the genetic string (*l*)
b) Population size (*N*)
c) Mutation probability (p_m)
d) Crossover probability (p_c)
e) Number of generations (*G*)
f) Seed for random number generator

Fitness of individual is calculated using following formula:

$$F(s) = (x/2^l)^{10},$$

where x is an integer representing genetic binary string.

Experiments consist of trying multiple combinations of parameters specified above. We also run experiment with each combination multiple times to achieve better results and statistics.

3. Experiment Flow

We were provided with base case parameters, assuming they are the most optimal. We will be changing one parameter at the time to see how overall picture will be changing.

Base case:

l = 20, N= 30, p_m= 0.033, p_c= 0.6, G = 10, seed = 1234

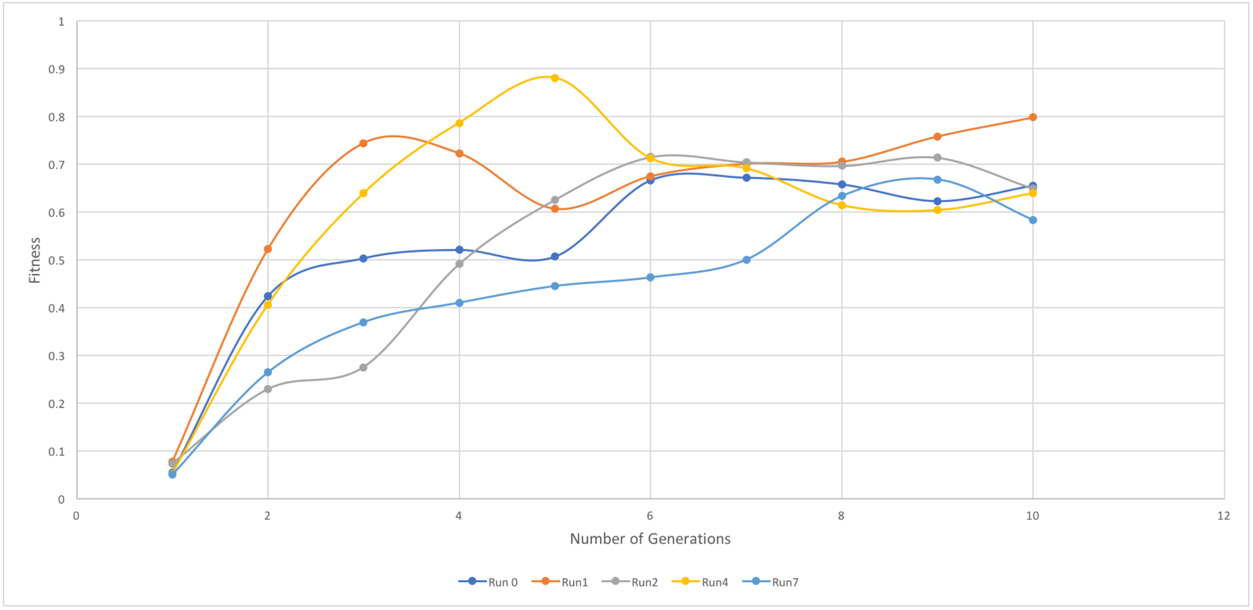


Figure 3. Average fitness

In Figure 3, we can observe how in all runs we approximately start with the same fitness value, although fitness varies a lot till ~7th generation, it approaches a higher value towards the end.

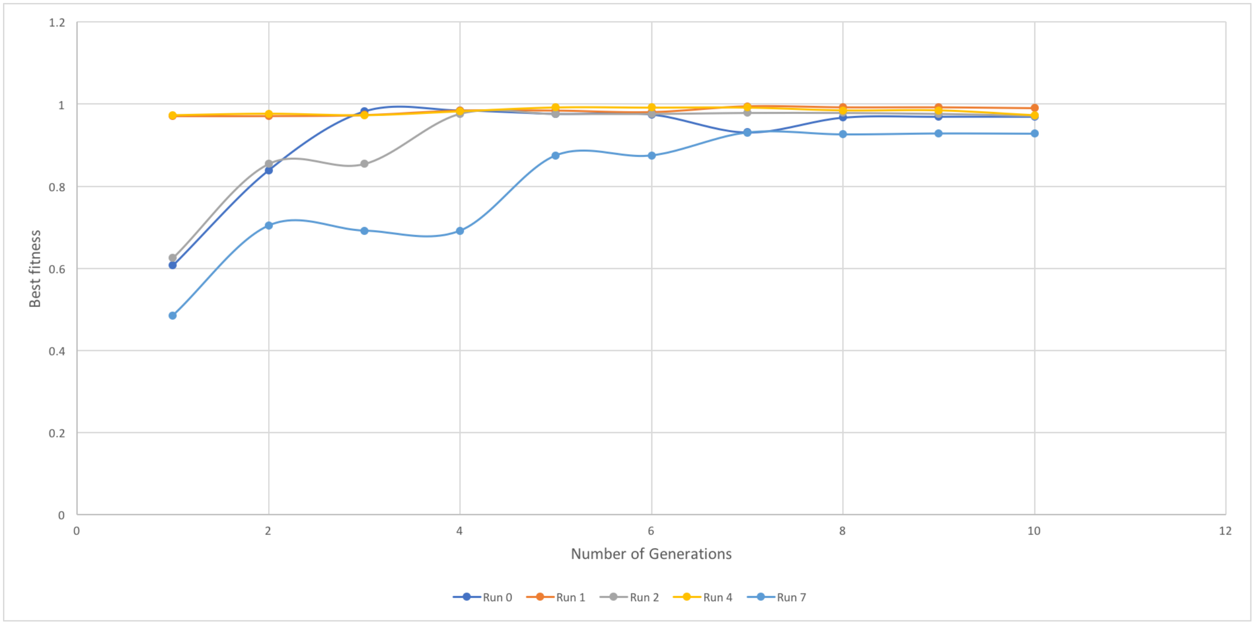


Figure 4. Best fitness

Figure 4 demonstrates fitness of the best "individual" across all generations. We can observe how value stays stable for Run 1 and 4, meaning that there was always a very fit "individual", in other runs the fittest value isn't that great but after crossover and mutation of population value changes.

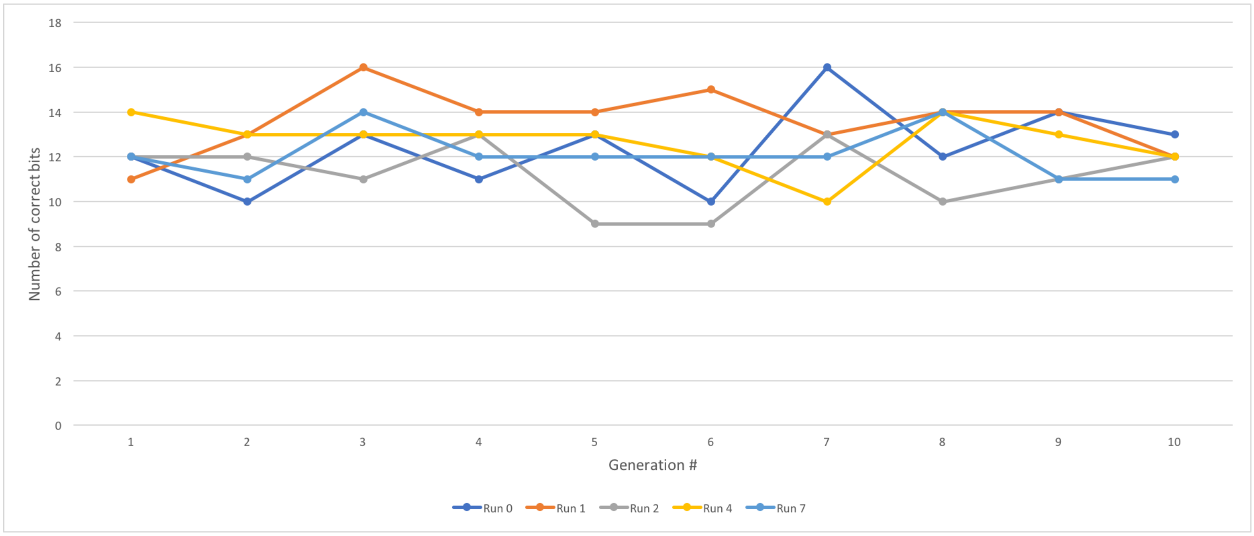


Figure 5. Number of correct bits

In figure 5, we prove that even with crossover of the fittest, mutation may still have a negative effect when number of correct bits actually starts to decrease. So, there is no guarantee that with each generation we will be getting only better genetic material.

I. Observing the length of genetic string

i. Combination 1

l = 5, N = 30, p_m= 0.033, p_c= 0.6, G = 10, seed = 1234

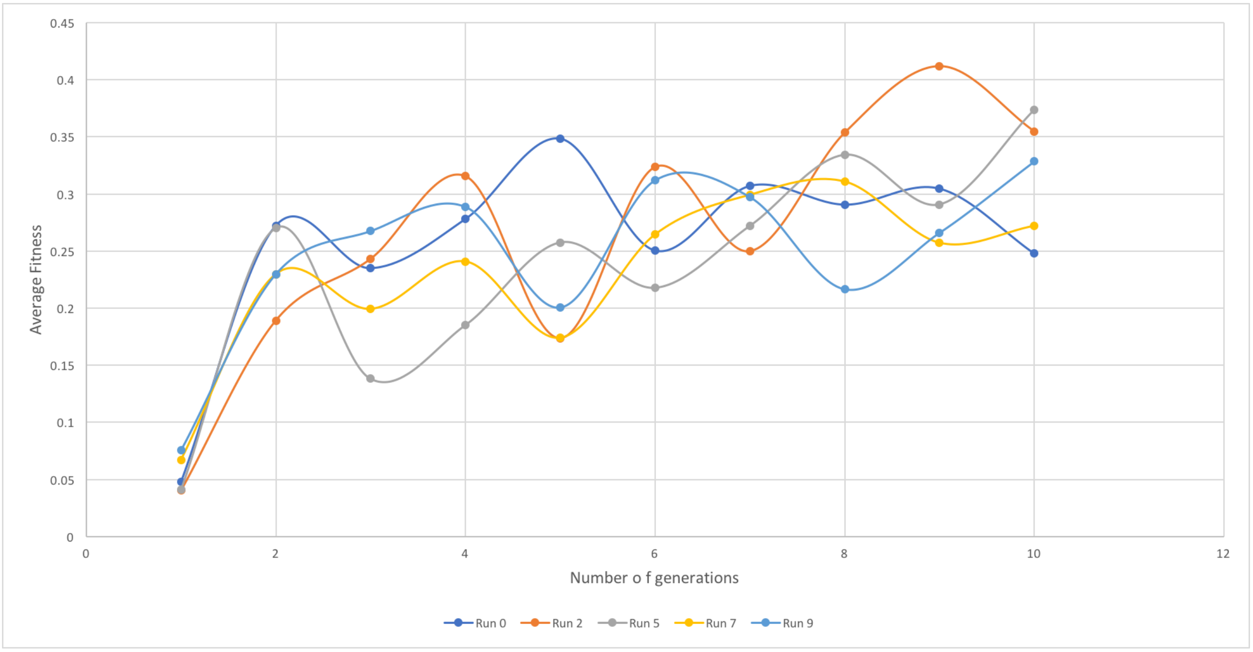
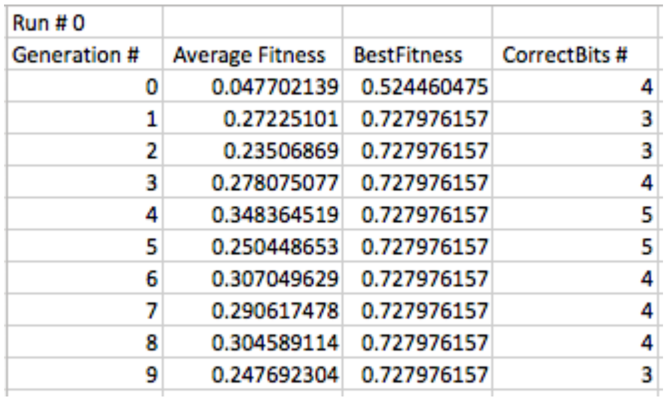


Figure 6. Average Fitness

For this combination of parameters, we observe a very low average fitness within 10 generations, it seems to be growing for 3 out 5 runs in Figure 6, but max value is still around ~0.43.



Run # 0			
Generation #	Average Fitness	BestFitness	CorrectBits #
0	0.047702139	0.524460475	4
1	0.27225101	0.727976157	3
2	0.23506869	0.727976157	3
3	0.278075077	0.727976157	4
4	0.348364519	0.727976157	5
5	0.250448653	0.727976157	5
6	0.307049629	0.727976157	4
7	0.290617478	0.727976157	4
8	0.304589114	0.727976157	4
9	0.247692304	0.727976157	3

Figure 7. Data table for Run 0

In the table above, we also can see that the best fitness is ~0.72 even when the number of correct bits is 5. The results are very similar for other 9 runs.

ii. Combination 2

l = 10, N = 30, p_m= 0.033, p_c= 0.6, G = 10, seed = 1234

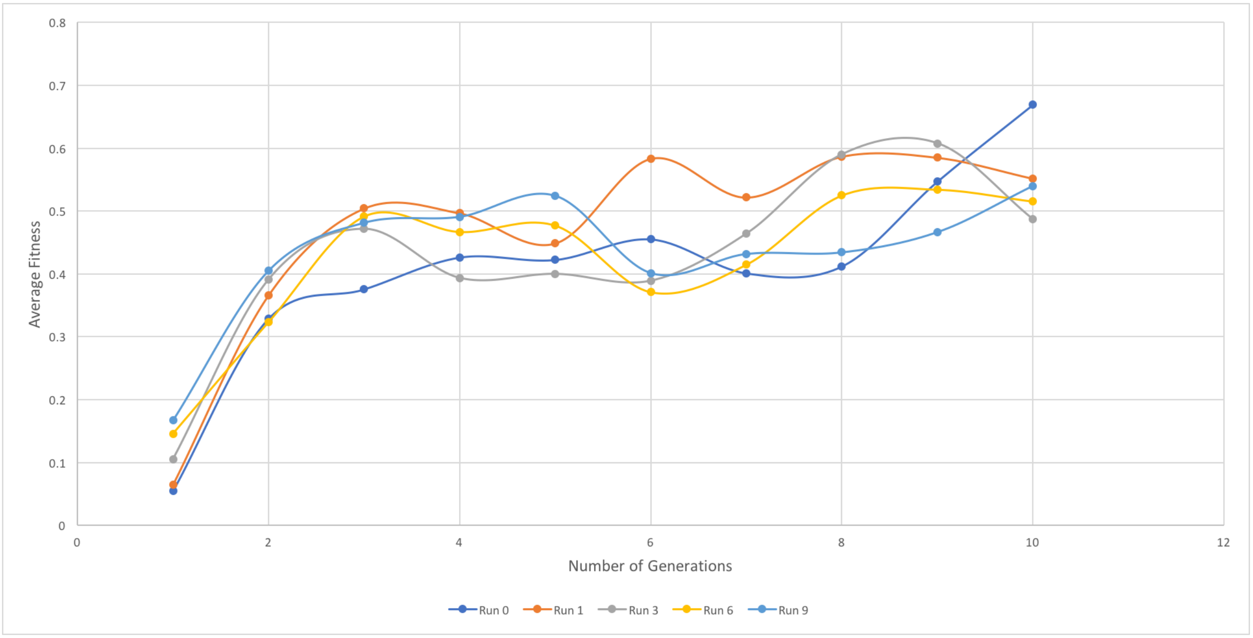


Figure 8. Average fitness

Increasing number of genes only by 5 units shows much better results.

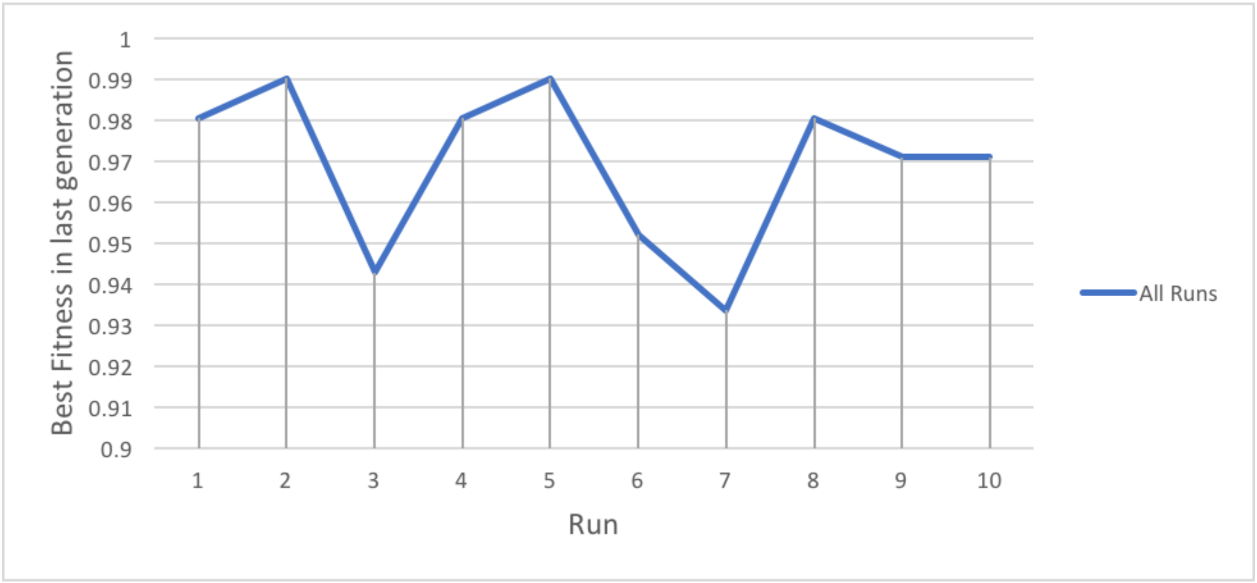


Figure 9. Last generation best fitness

Looking at the last generation in all runs (Figure 9) , the best fitness varies from ~0.93 – 0.99, which is also a lot better than in previous example.

iii. Combination 3

l = 30, N = 30, p_m= 0.033, p_c= 0.6, G = 10, seed = 1234

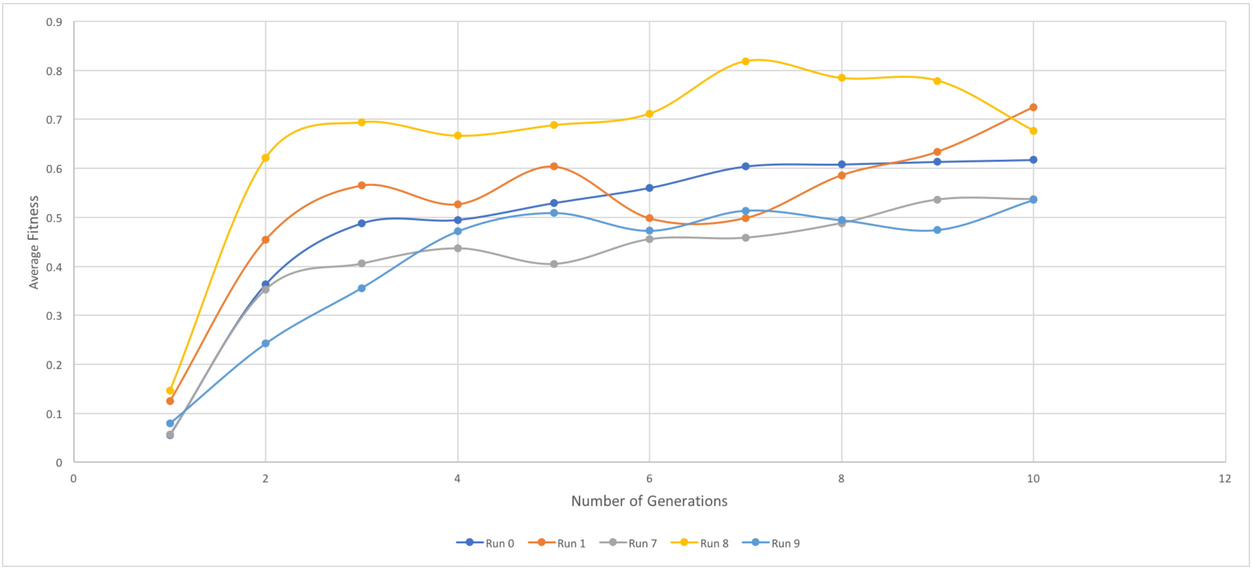


Figure 10. Average fitness

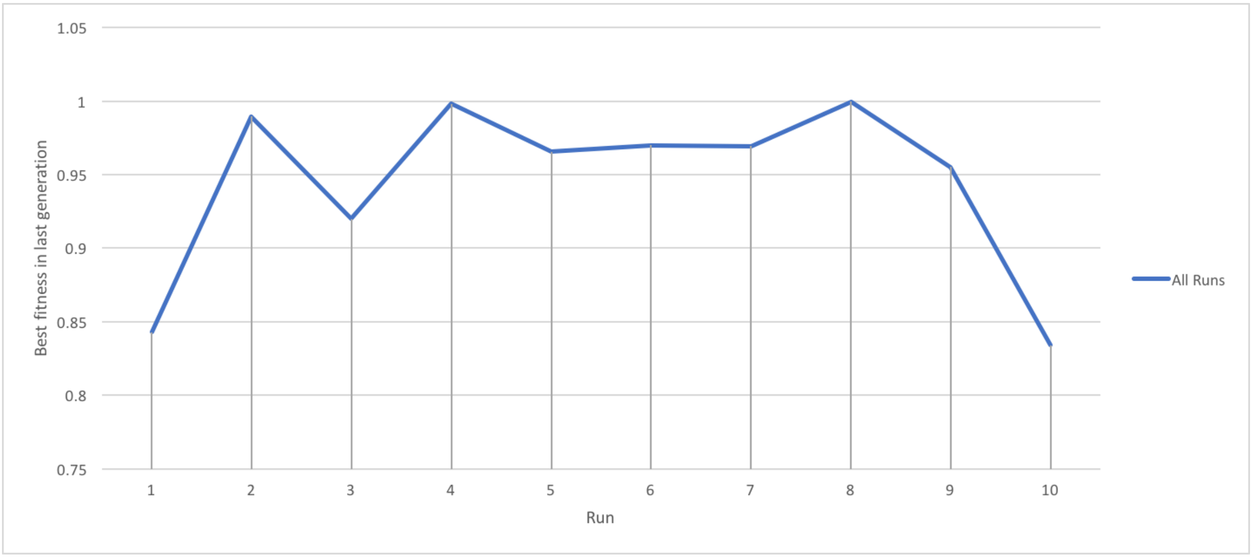


Figure 11. Best fitness in last generation

The average fitness seems to be slightly better than in previous combination, but it is worse than in base case, which is considered to be optimal. Also best fitness in last generations seems to vary much more and the lowest value hits ~0.83, which is worse than in even previous example.

iv. Combination 4

l = 40, N = 30, p_m= 0.033, p_c= 0.6, G = 10, seed = 1234

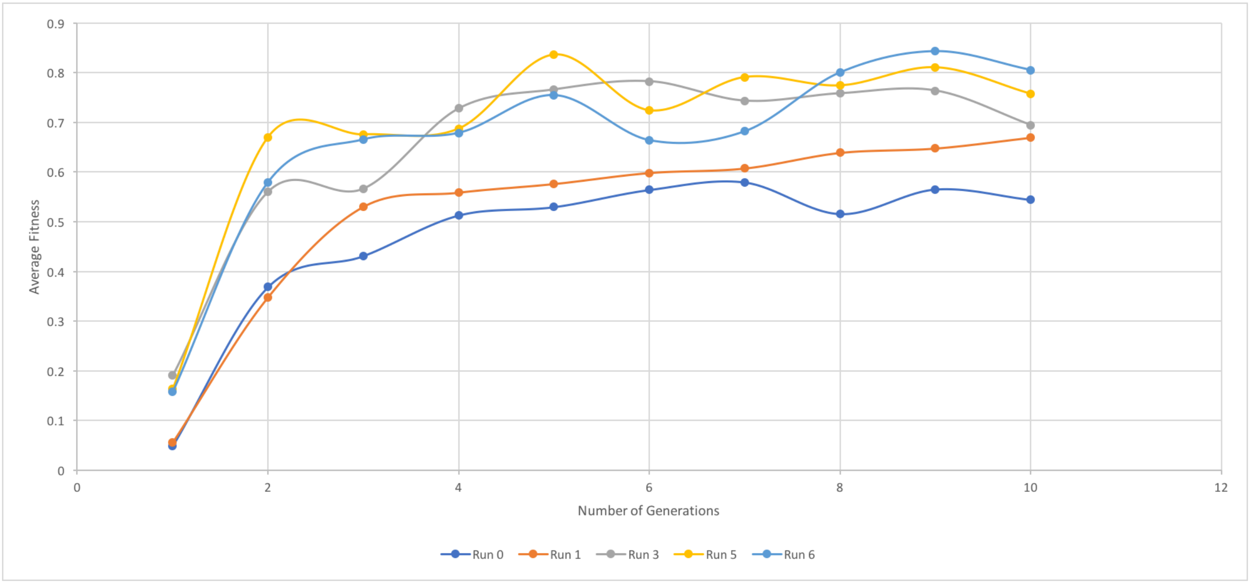


Figure 12. Average fitness

In this example, average fitness seems to be more scattered, it probably means that more with more generations data would be more consistent for such a large number of genetic bits.

Best fitness values are similar to example above.

II. Observing the size of population

i. Combination 1

l = 20, N = 10, p_m= 0.033, p_c= 0.6, G = 10, seed = 1578

In a picture below, we can see results that show how inconsistent data is when population size small. Fitness seems to grow with each generation, but I guess is more a matter of luck of how initial strings get generated, things can have really good or bad, since there is not that much of a randomness.

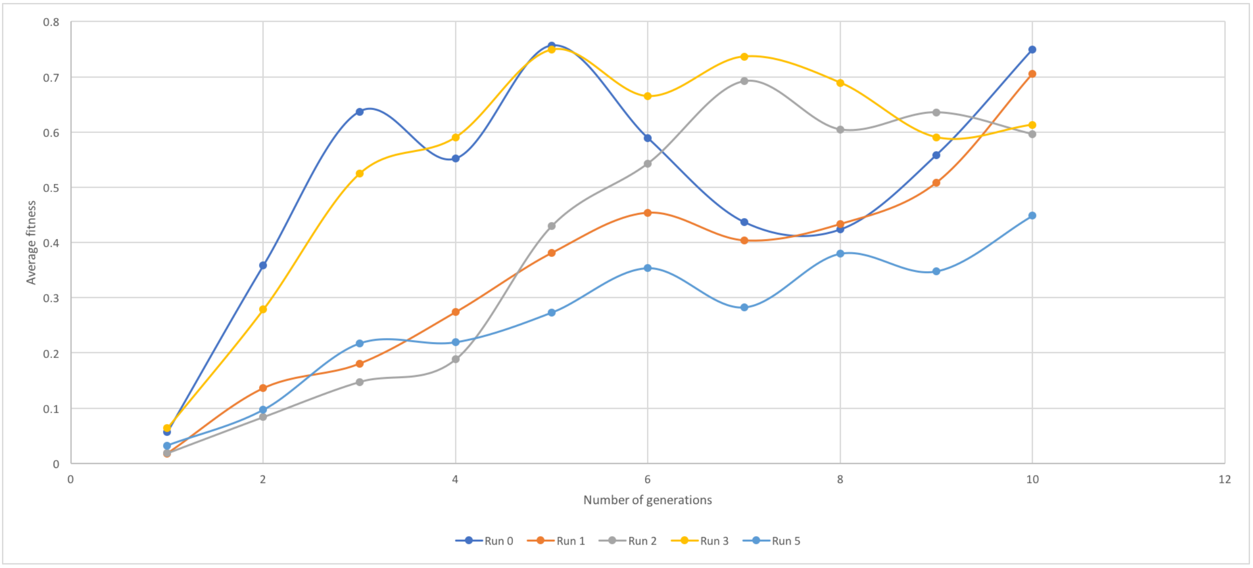


Figure 13. Average fitness

ii. Combination 2

l = 20, N = 20, p_m= 0.033, p_c= 0.6, G = 10, seed = 1578

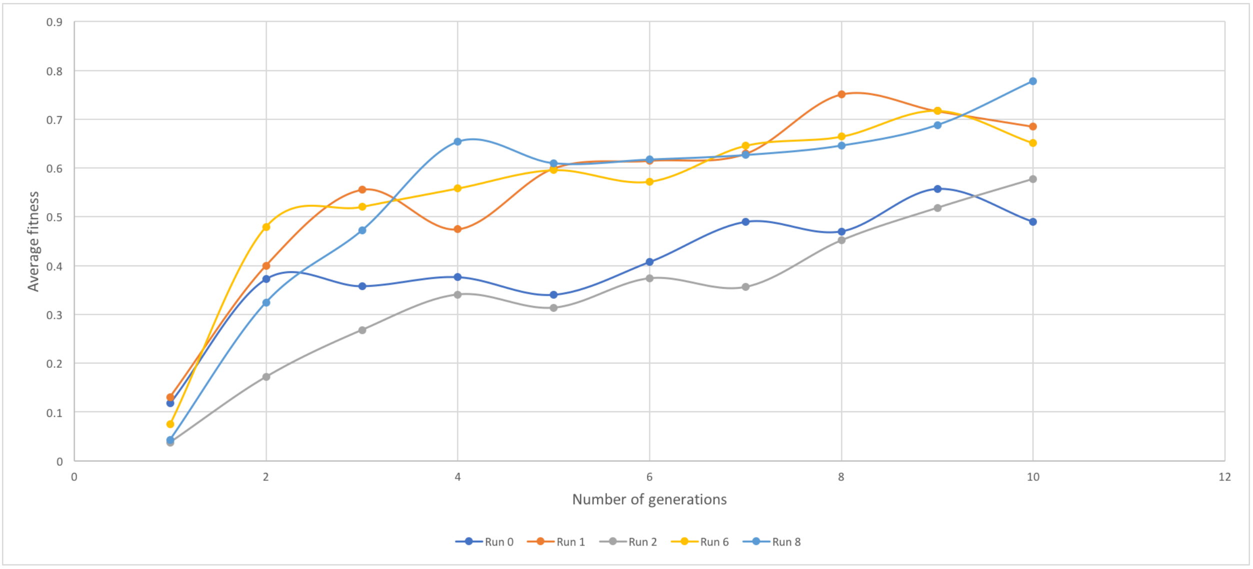


Figure 14. Average fitness

Increasing population size helps us get better randomness and smooth results. In almost all runs, best fitness in last generation hit 0.99.

iii. Combination 3

l = 20, N = 40, p_m= 0.033, p_c= 0.6, G = 10, seed = 1578

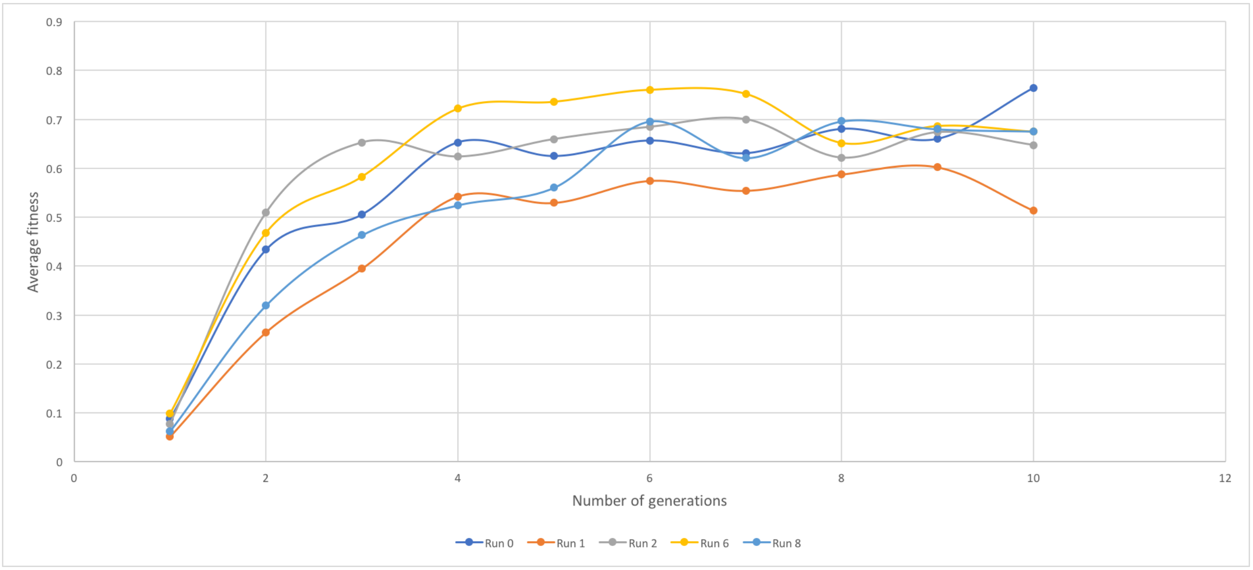


Figure 15. Fitness Average

This is probably the most consistent result for average fitness so far, let's look at the best fitness.

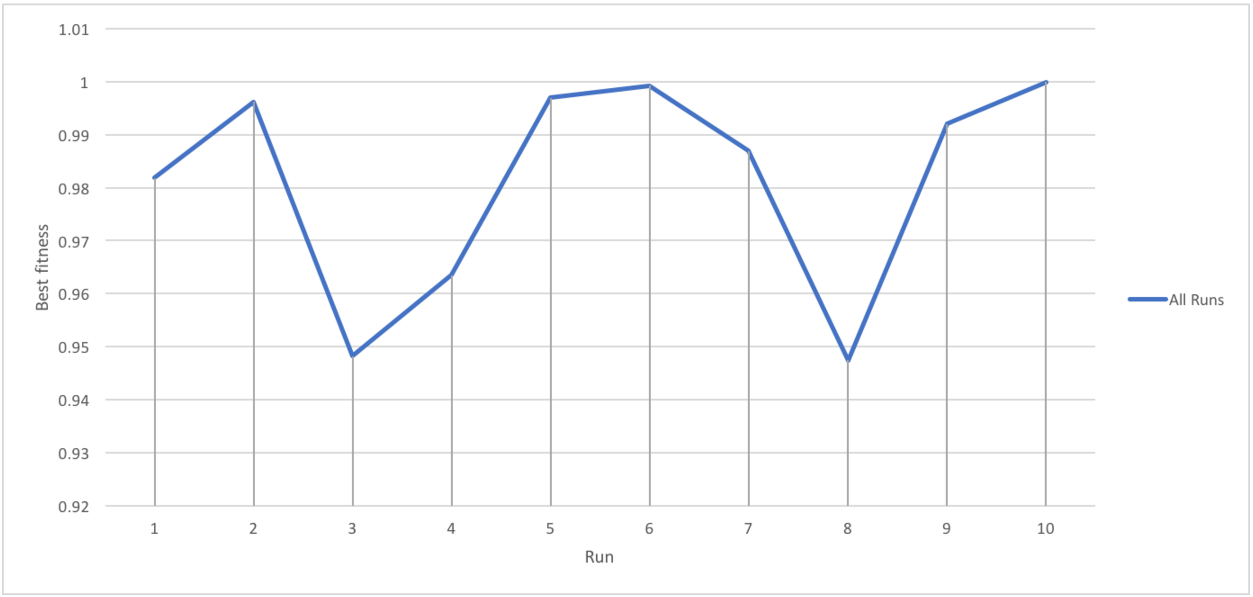


Figure 16. Best Fitness

Best fitness is always over 0.93, which is also pretty good. Nevertheless, average fitness seems to be below 0.8, which is worse than in optimal combination explored at the beginning.

iv. Combination 4

l = 20, N = 50, p_m= 0.033, p_c= 0.6, G = 10, seed = 1578

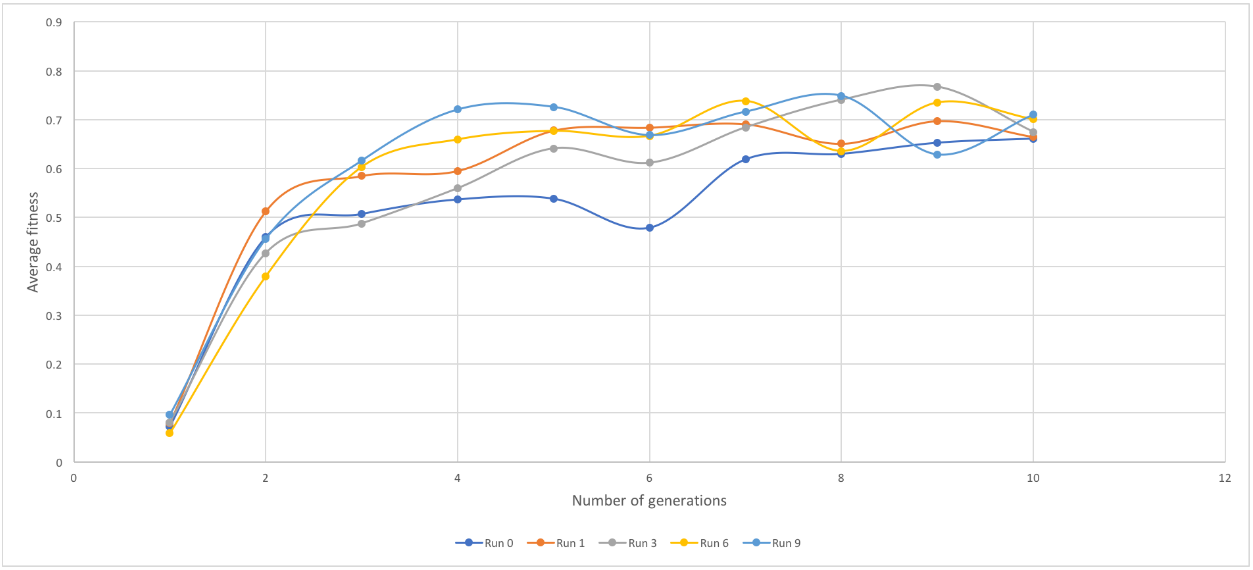


Figure 17. Average fitness

This example concludes that with higher population size we get smother data and better randomness which allows population to mate and mutate evenly over time.

The best fitness is also ~0.99 in last generation in almost all runs.

III. Observing number of generations

We are not going to observe smaller number of generations, because intuitively it should show lower fitness results.

i. Combination 1

l = 20, N = 30, p_m= 0.033, p_c= 0.6, G = 20, seed = 1432

In Figures below we see picture similar to the base case, only graph is longer. We expected the value of fitness to grow with generation, but as we can see, this is not the fact. Also best fitness in last generation seems to be better a little.

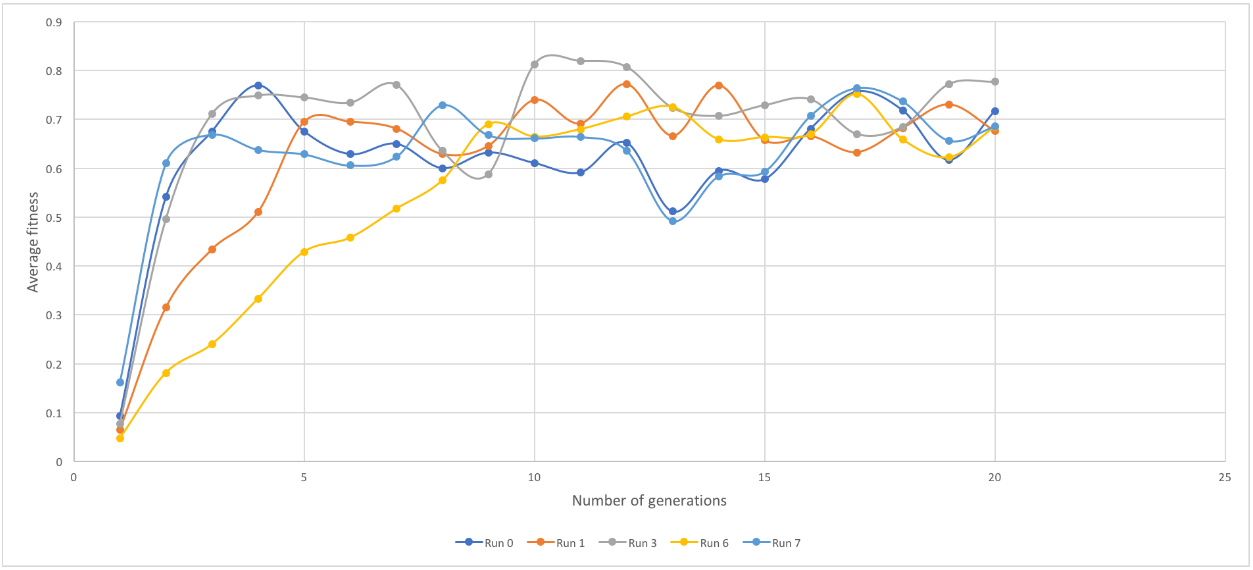


Figure 18. Average fitness

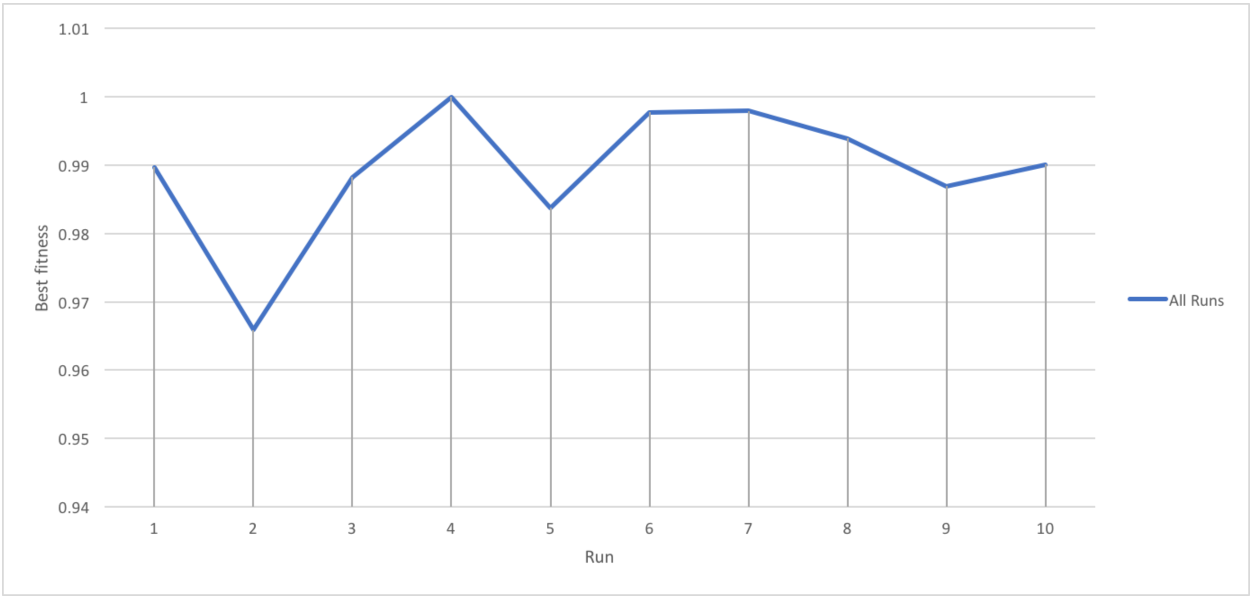


Figure 19. Best fitness

And let's double number of generations.

ii. Combination 3

l = 20, N = 30, p_m= 0.033, p_c= 0.6, G = 40, seed = 1432

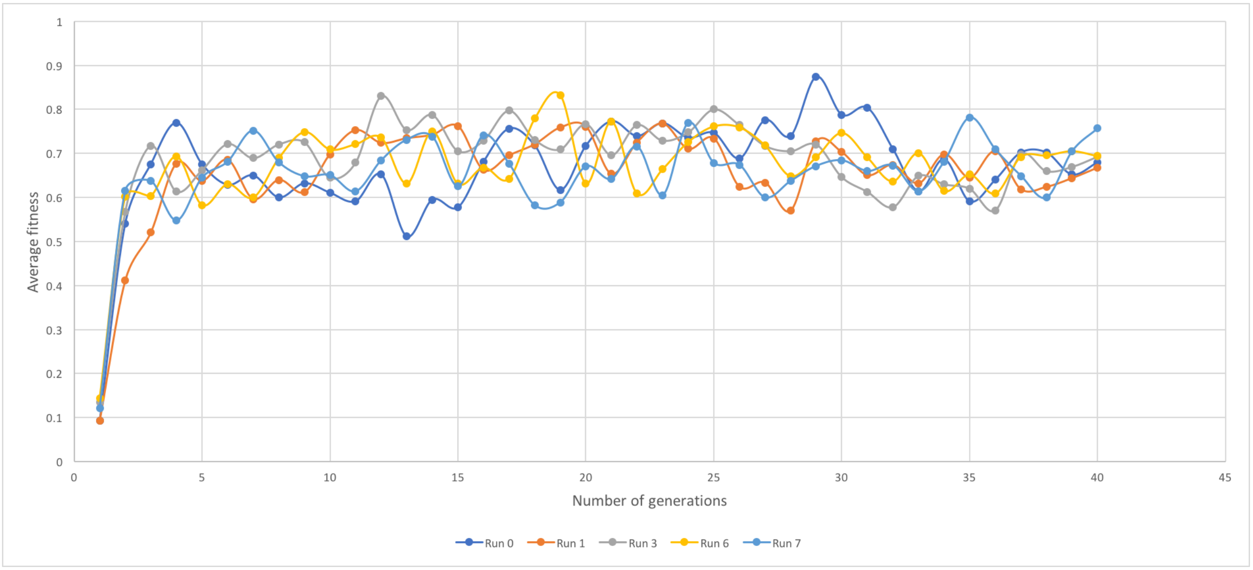


Figure 20. Average fitness

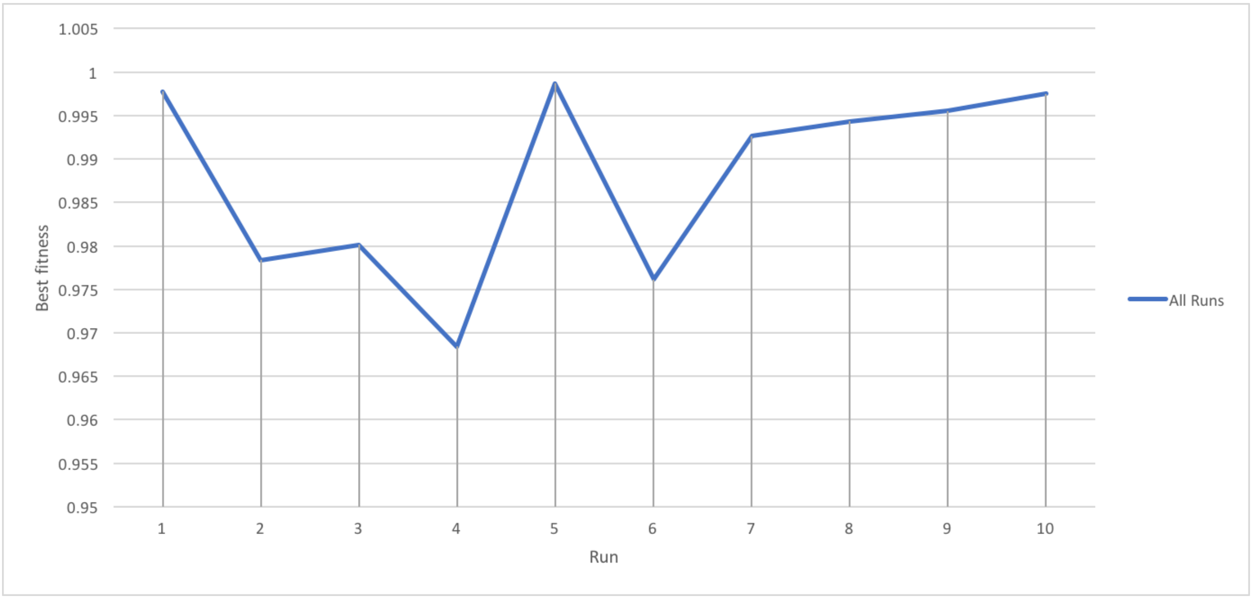


Figure 21. Best fitness

What we can observe from this combination of parameters is that data continues to be slightly inconsistent, but best fitness increased among last generations. I almost all examples it hits 0.99.

IV. Observing mutation probability

i. Combination 1

l = 20, N = 30, p_m= 0.015, p_c= 0.6, G = 10, seed = 2012

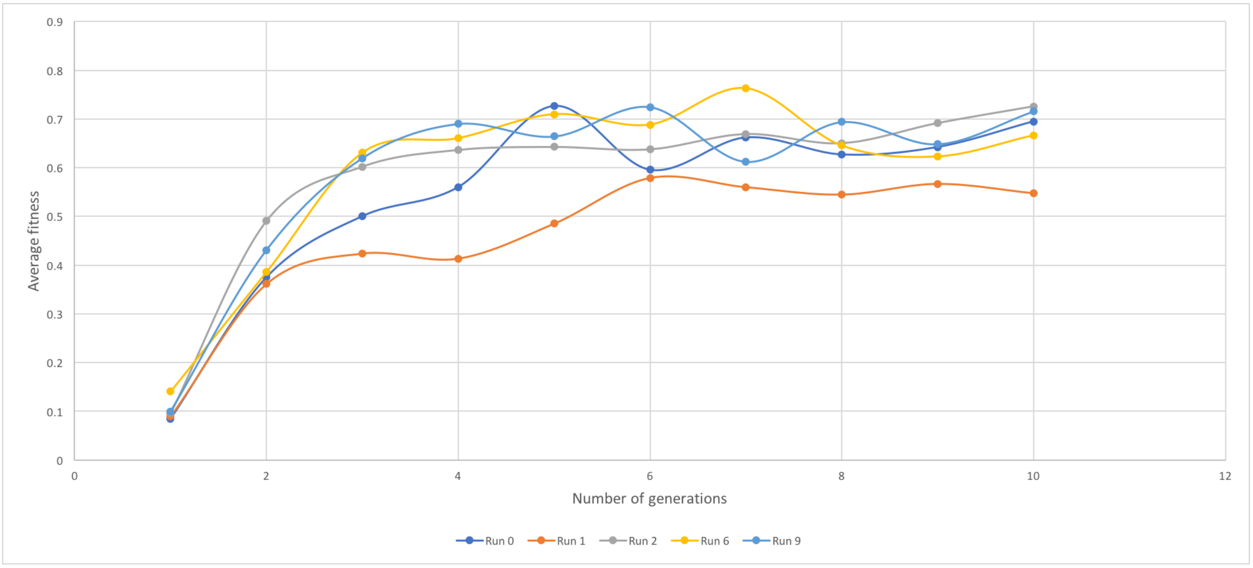


Figure 22. Average fitness

Best fitness values among last generations vary from 0.95- 0.99.

ii. Combination 2

l = 20, N = 30, p_m= 0.063, p_c= 0.6, G = 10, seed = 651

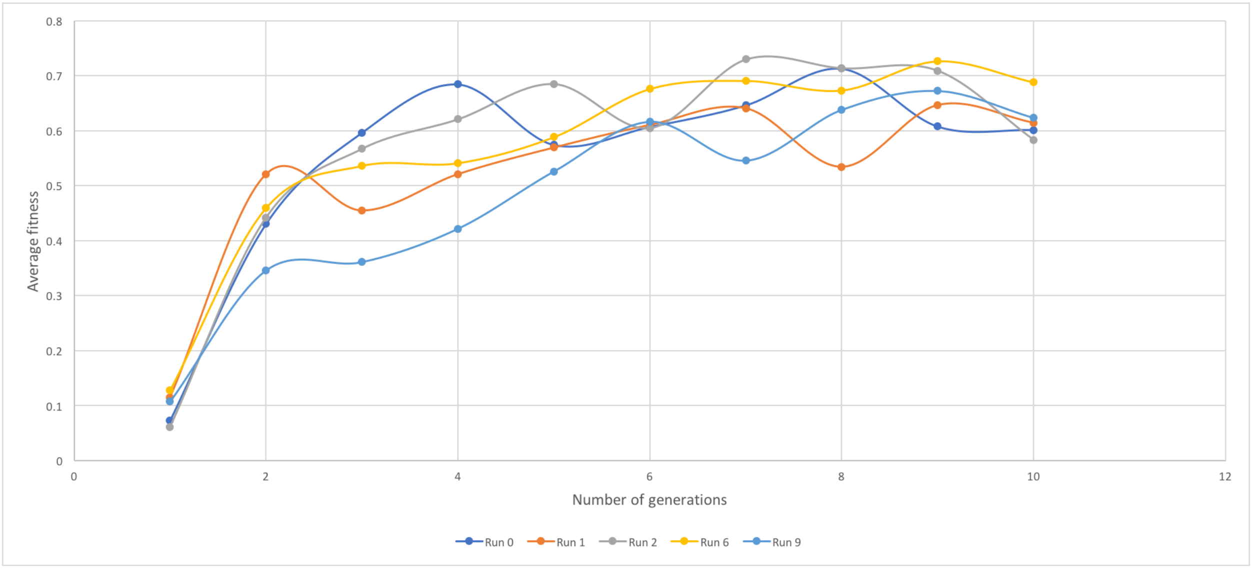


Figure 23. Average fitness

Lower mutation rate seems to have better fitness average towards the end of generations. But as it was stated before, too much of mutation is never good.

V. Observing crossover probability

In this experiment, we will also look at two values, higher an lower ones.

i. Combination 1

l = 20, N = 30, p_m= 0.033, p_c= 0.1, G = 10, seed = 531

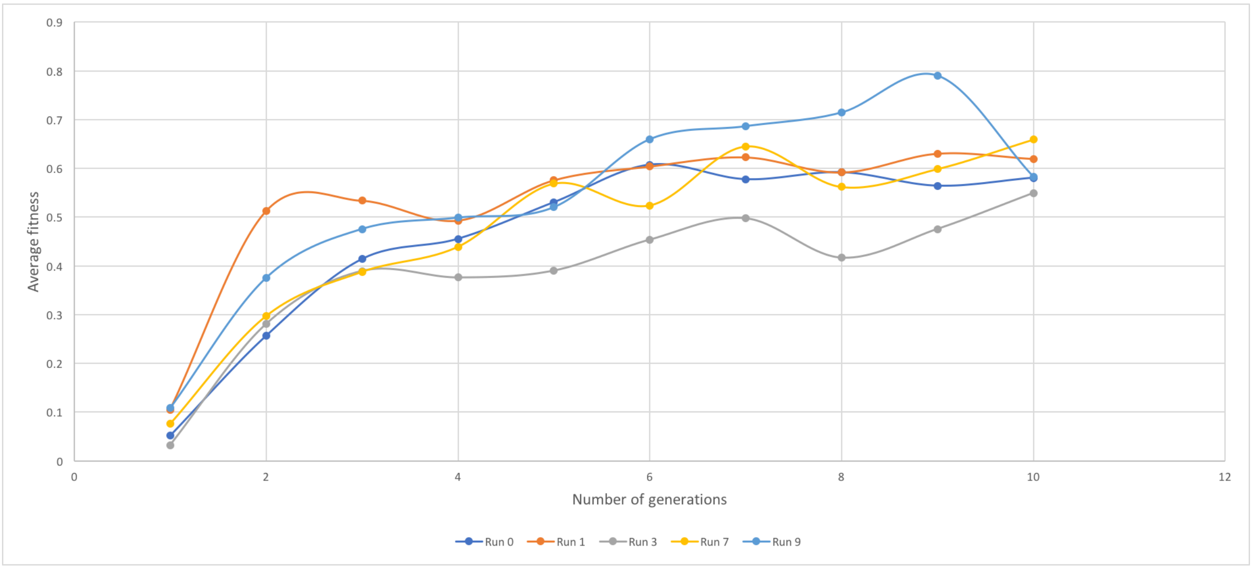


Figure 24. Average fitness

The result seems to be pretty constant, except for that 9th generation, let's look at higher value, since it is really hard to conclude anything. Also, best fitness among last generations is varying between 0.69-0.99, which unusual.

ii. Combination 3

l = 20, N = 30, p_m= 0.033, p_c= 0.9, G = 10, seed = 635

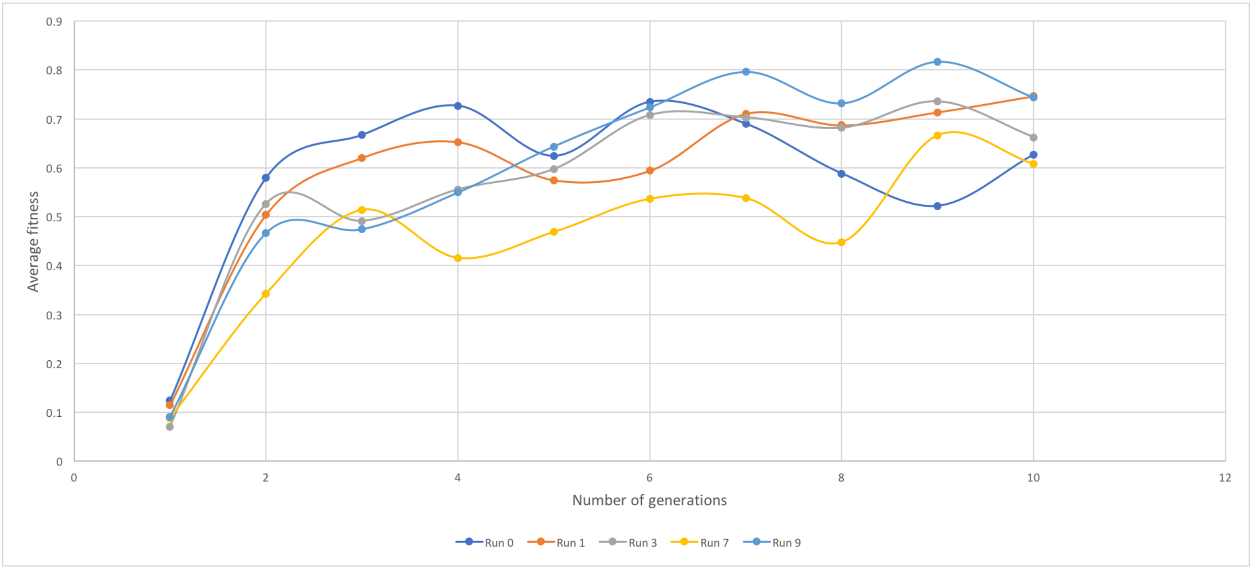


Figure 25. Average fitness

This graph looks very similar to the one above, which makes me think that we have crossover probability just as another way of adding randomness, but results don't change much form what value we get to choose.

4. Conclusions

Finishing the project, I would want to conclude that better and more consistent results were achieved when population size was pretty large like 40 or so, also having more gens in genetic string adds more diversity to the population. Having more generations increases fitness results at the very low fraction, but that's how it would be in a real life. We don't have each next generation twice taller and prettier in real life. Probability parameters don't play large of a role in at least my experiment, mutation probability has a very slight effect on average fitness and number of correct bits in last generations. To see how much my assumptions are wrong or write I ran final experiment for l = 30, N = 50, p_m= 0.033, p_c= 0.6, G = 300.

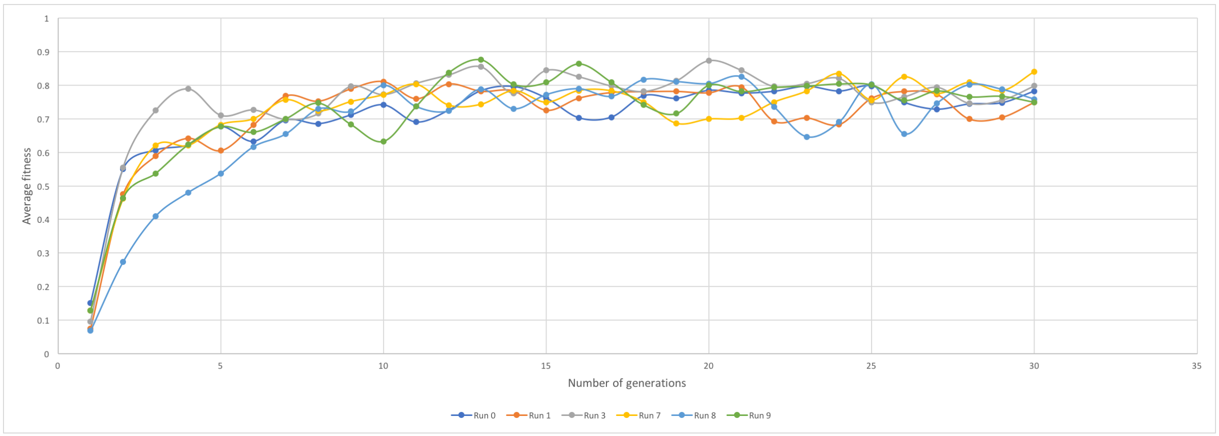


Figure 26. Final average fitness

The result seems pretty consistent, and all best fitness values in last generations are at least 0.99.

Literature Cited

Flake, Gary William. *The Computational Beauty of Nature*. Cambridge, MA: The MIT Press, 1998. Print.