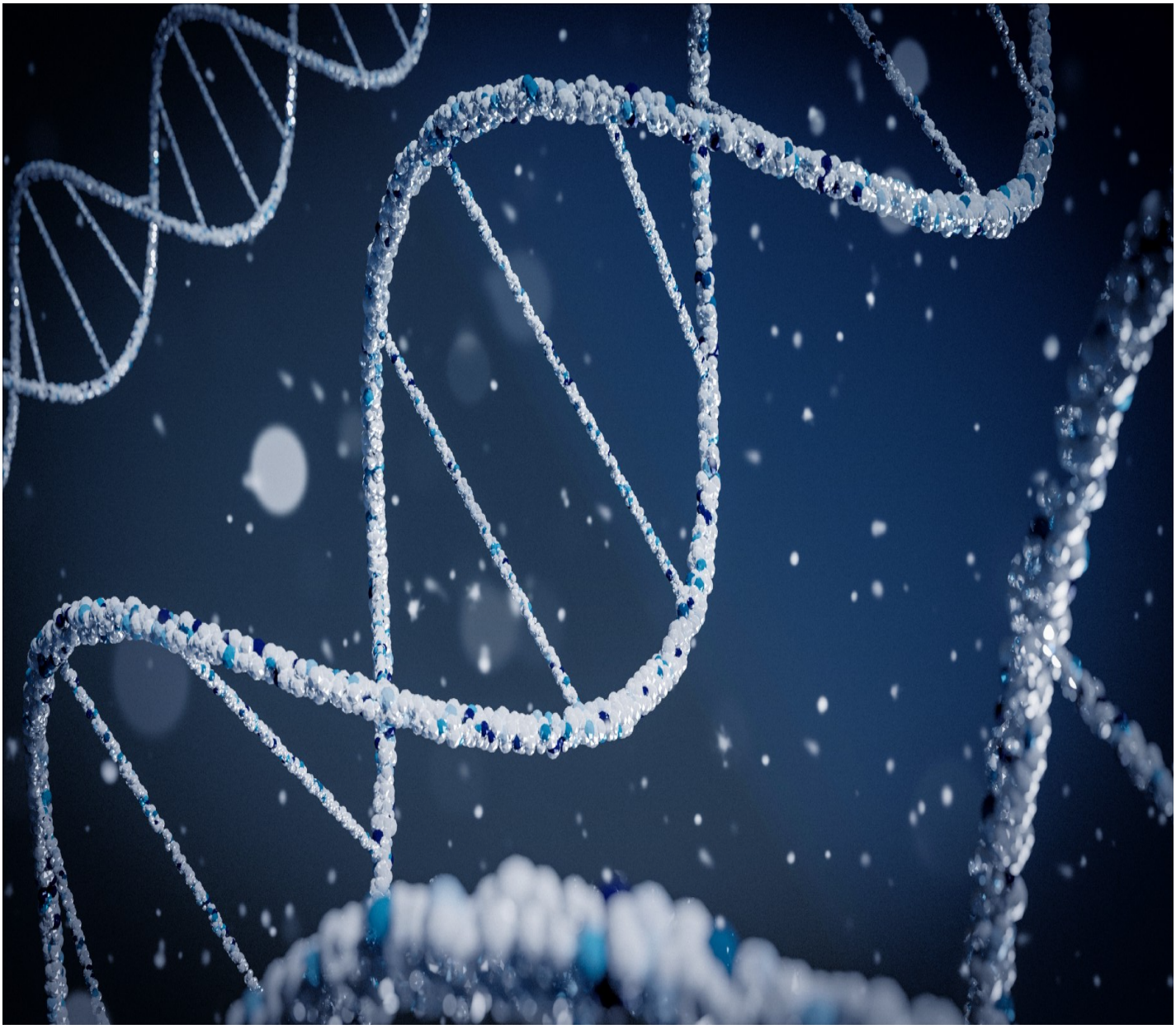
Report

Genetic Algorithm



Cristian Sirbu

Table of Content

Overview

The Genetic Algorithm is a method of optimization based on natural genetics and natural selection mechanics. To develop search and optimization procedures, the Genetic Algorithm uses the principles of natural genetics and natural selection. It is used to find a near-optimal solution in a short amount of time. The representation of a genetic algorithm is done with sub-chromosomes of varying lengths. GA was created to find the best order-scheduling solution possible. It is a tool used to optimise process variables in a variety of processes. The genetic algorithm is a multi-path algorithm that searches multiple peaks in parallel, reducing the risk of local minimum trapping and solving multi-objective optimization problems.

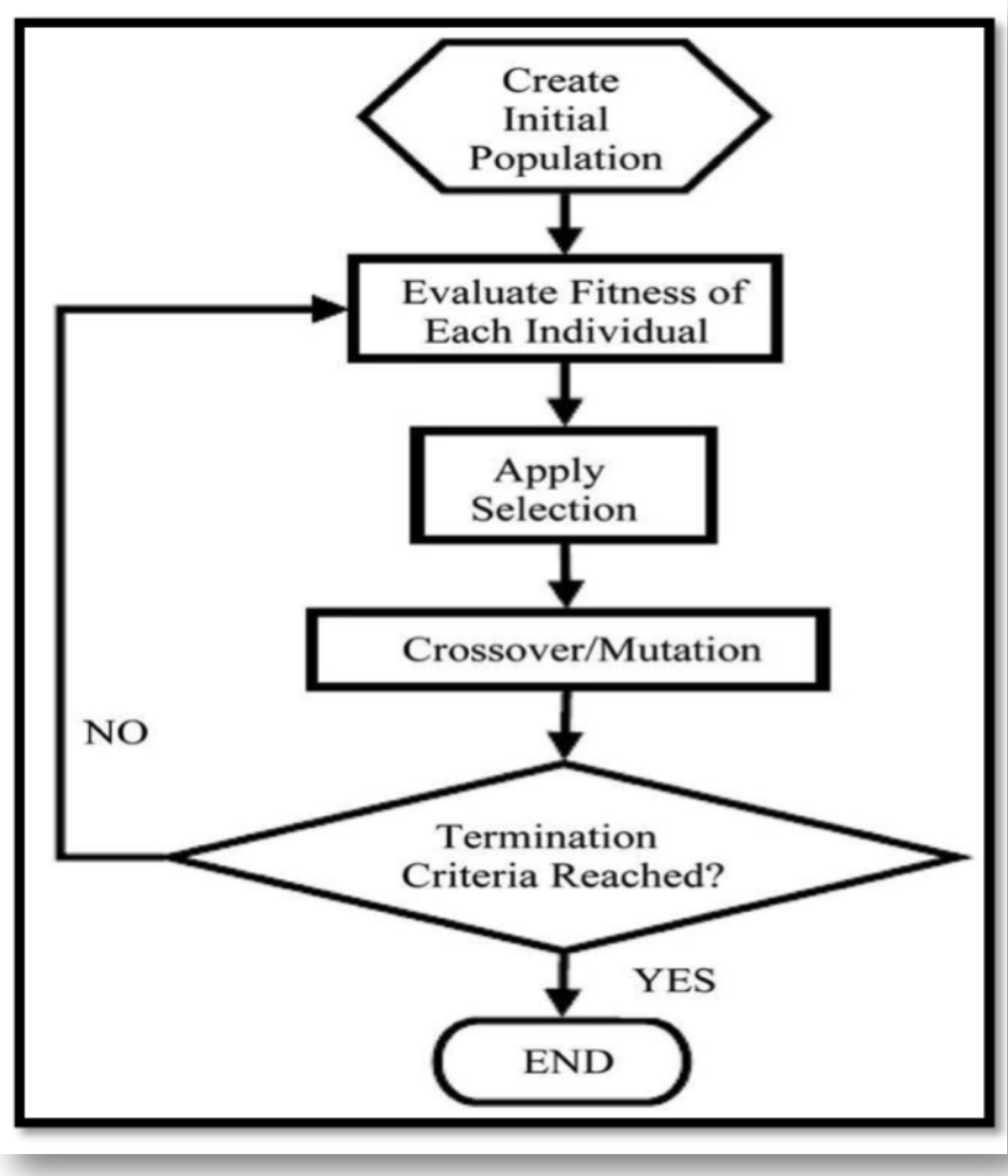
Introduction

Genetic algorithms (GAs) are general-purpose, stochastic, iterative, evolutionary search strategies based on population genetics and natural selection principles. Genetic Algorithms are well-known heuristics that have been used to solve a wide range of optimization problems, thereby validating the mathematical model's consistency. Holland, who was interested in natural adaptive systems and their simulation in software, was a forerunner in the development of GAs. GAs have primarily been used in optimization and machine learning. They have been used in diverse fields, including the social sciences, management science, biology, and engineering.

Genetic algorithms are a form of evolutionary algorithm. These are algorithms based on natural evolution principles, and they are classified into four types: genetic algorithms, genetic programming, evolution strategies, and evolutionary programming.
These algorithms are all based on a population of individuals. The genetic algorithm is composed entirely of genetic operators such as reproduction, crossover, and mutation. The algorithms' applicability is broad because they perform well in a range of problems where traditional methods fail, and Goldberg distinguishes genetic algorithms from classical approaches by four features.

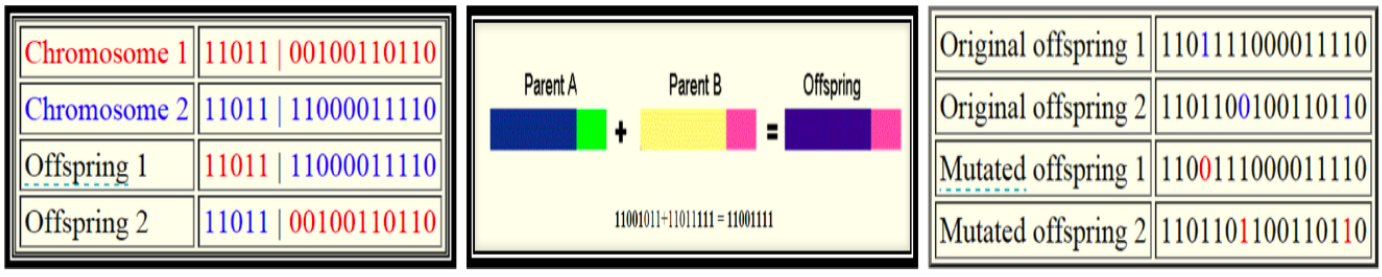
In this report, I'll look at the knapsack issue. A P vs NP problem is an example of how a genetic algorithm is used.

How They Work



1. Initialise random pop P
2. Associate each individual with a position x
3. Pick one individual at random, i.e., genotype G_1 at position x_1
4. Pick a second individual G_2 in the local neighbourhood of the first, i.e., pick a competitor from the local neighbourhood in the range **(x_1+1)** to **(x_1 + k) (start with k=2)**
5. Compare G_1 and G_2 finding a winner (W) and loser (L)
6. Copy each gene of the winner W to the L with crossover probability (Pcrossover, say 0.5 to start)
7. Add a mutation to the L and insert it back to the population.
8. Until success or give up, go to 3.

The knapsack (KP) problem is a classic combinatorial optimization problem that has been studied for over a century. And humanity does not know if there is an algorithm that runs swiftly enough. A GA employs a population of potential solutions. That would be combinations of goods in our backpack in our scenario. The population is binary and randomly created at the start, and it can be of any number, from a few individuals to thousands. As a result, we begin our evolutionary process in complete chaos. After then, each member of the population is examined, and a fitness value is calculated based on how well it fulfils our desired requirements. We empower Natural Selection to increase the general fitness of our population by retaining only the best individuals for the following generation. We create new individuals during crossover by merging aspects of our chosen individuals. The idea is that combining characteristics from two or more people will be able to generate a more 'fit' child who will inherit the best traits from each of its parents. Mutations are the final step in the evolution of a generation. By introducing a small amount of randomness into our populations' genetics, we exclude the possibility that the created children may be gained from the first generation.



(a) (b) (c)

Performance

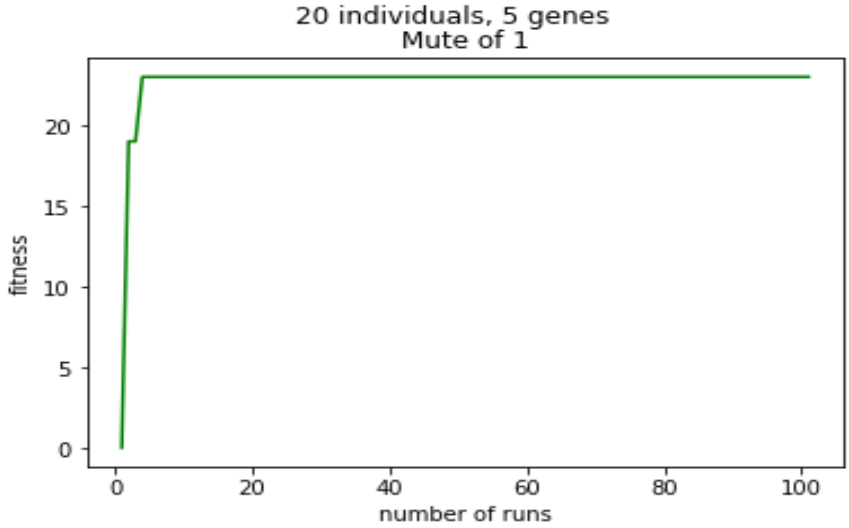
Table 1. The GA parameters, PS - population size, CP - crossover probability, MP - mutation probability, used in previous works.

PS	CP	MP
30	0.95	0.01
80	0.45	0.01
100	0.9	1
100	0.8	0.005
50	0.9	0.03
50	1	0.03
50	0.8	0.01
15	0.7	0.05
100	1	0.003
30	0.8	0.07
100	0.8	0.01
500	0.8	0.2
40	0.6	0.1
100	0.8	0.3
100	0.6	0.02

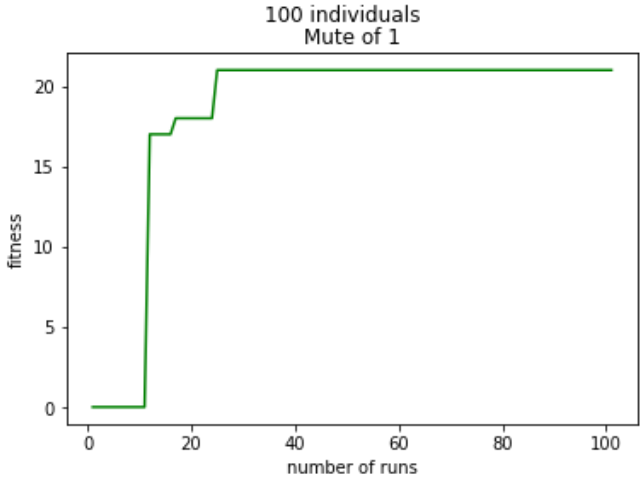
PS	CP	MP
30	0.6	0.05
76	0.8	0.05
4000	0.5	0.001
30	0.6	0.001
50	0.8	0.2
50	0.9	0.1
30	0.9	0.1
20	0.8	0.02
50	0.9	0.01
20	0.9	0.3
330	0.5	0.5
100	0.8	0.1

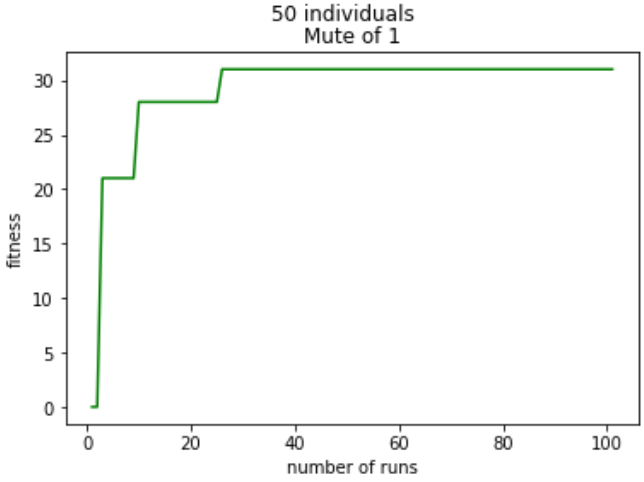
In this section, I will investigate how a genetic algorithm works. Because of the tiny population size, sample sizes are insufficient, resulting in poor solutions. A large population size denotes great search efficiency, which is preferable to a small population size, which also indicates a slow rate of convergence. Low crossover rates result in low exploration rates. For effective implementation of (GA), values ranging from (0.1–1) were to be chosen between very high and low crossover rates. High mutation rates usually resulted in a random search, and the mutation rates were invariably beneath the crossover rates, which were in the (0.01–1) range. The GA parameters used to implement GA are listed in **Table 1**.

In this experiment I will show how a GA performs in different instances and with different parameters. I will use this algorithm on an NP type of problem called Knapsack problem.

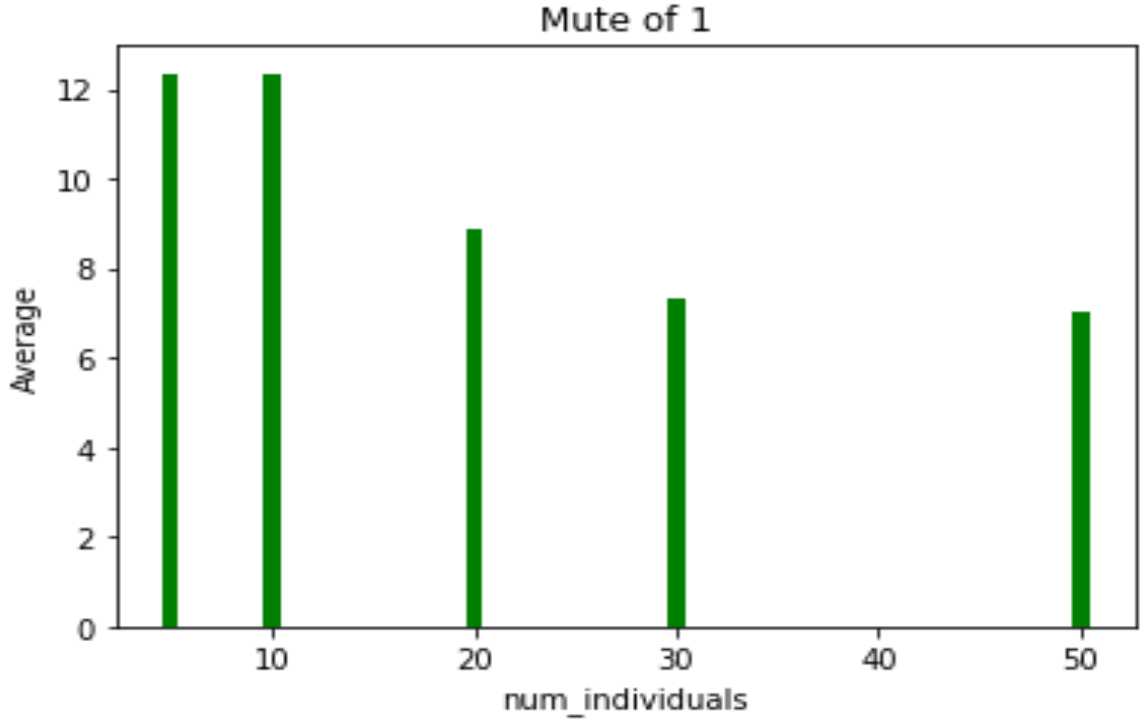


I started with a small population of 20 individuals by 5 genes and with only one mutation. As we can see after a couple of runs the fitness level does not change, so I changed a bit the parameters.

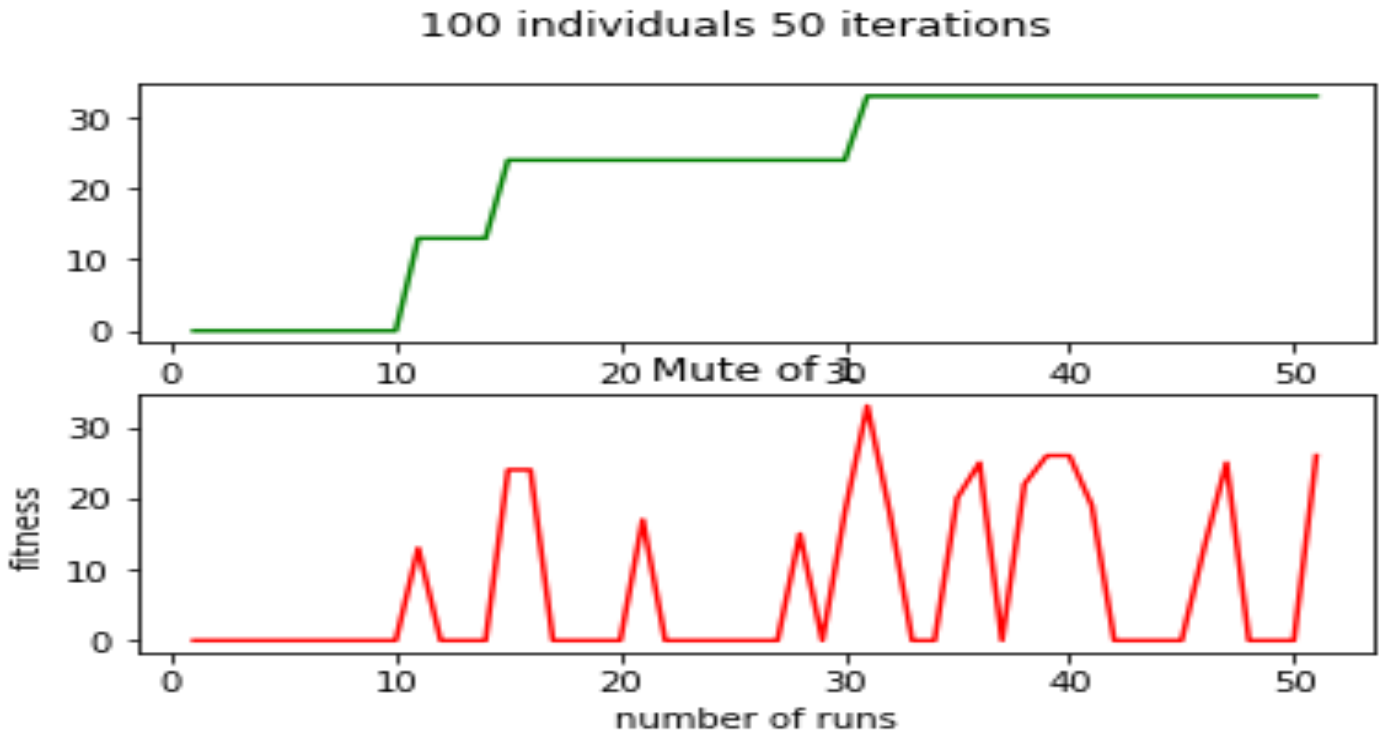




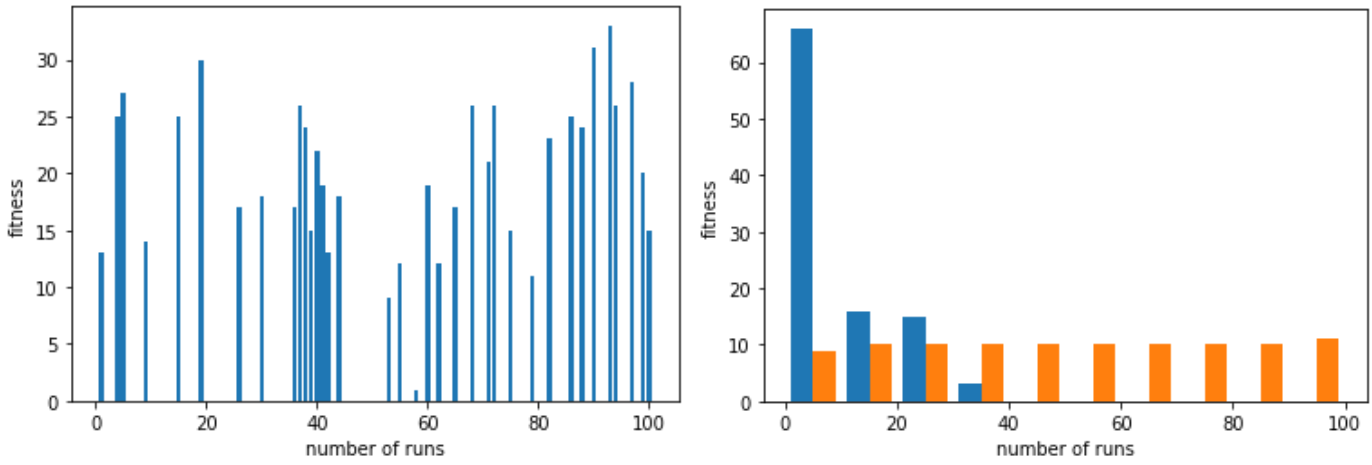
In this case I performed same 100 iterations but with a bigger population and a bigger sample of genes. The left plot has a population of 100 individuals while the right plot has a population of 50 individuals, but the gene number is the same, 10 genes. These plots are showing that this algorithm evolves by time and after a certain point of iteration it stagnates because it reached the best fitness and found the best combination of items. The problem is that we can't be sure that the last performance is the best one, but we can see in the plots how the fitness of algorithm evolves in time. Beneath I plotted the population size with its average fitness value. It shows that smaller population perform better than bigger one but in terms of calculations the complexity of a genetic algorithm reaches a point where time complexity is not anymore, a problem for big populations. Also, in this plot shows that we might not have the best solution at the end of the algorithm, but we have a decent one which corresponds to requirements.



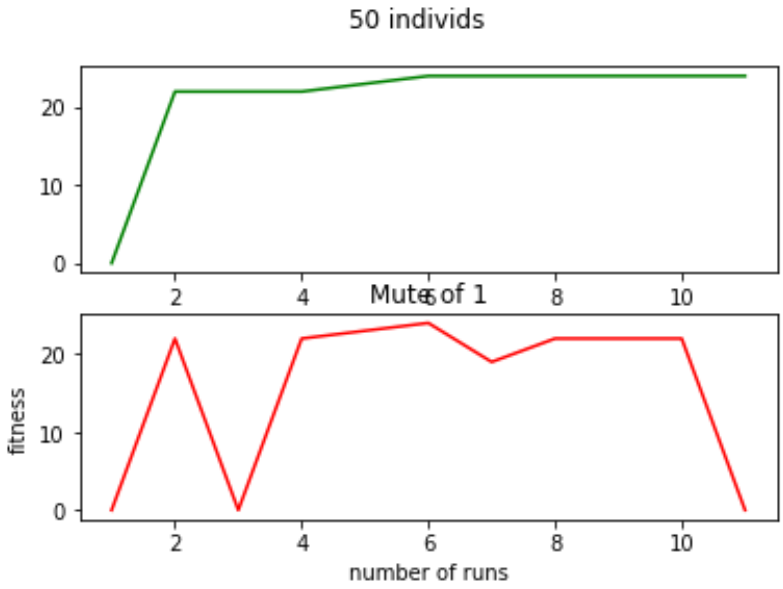
We can see the same things below but with a population of 100 individuals and 50 iterations, lowering the number of iterations doesn't let us unsolved. The second plot shows us how efficient are mutations regarding fitness of a generation. At the 50th run we ended the cycle with a good fitness but as we can see it's not the best one. The generation with best fitness is 30.



The randomness in evolution called "Natural Selection" applied to this algorithm prove us that in time we will get a good enough offspring by saving the predecessors with best genes and randomly intervene and mutate 1 gene per turn.

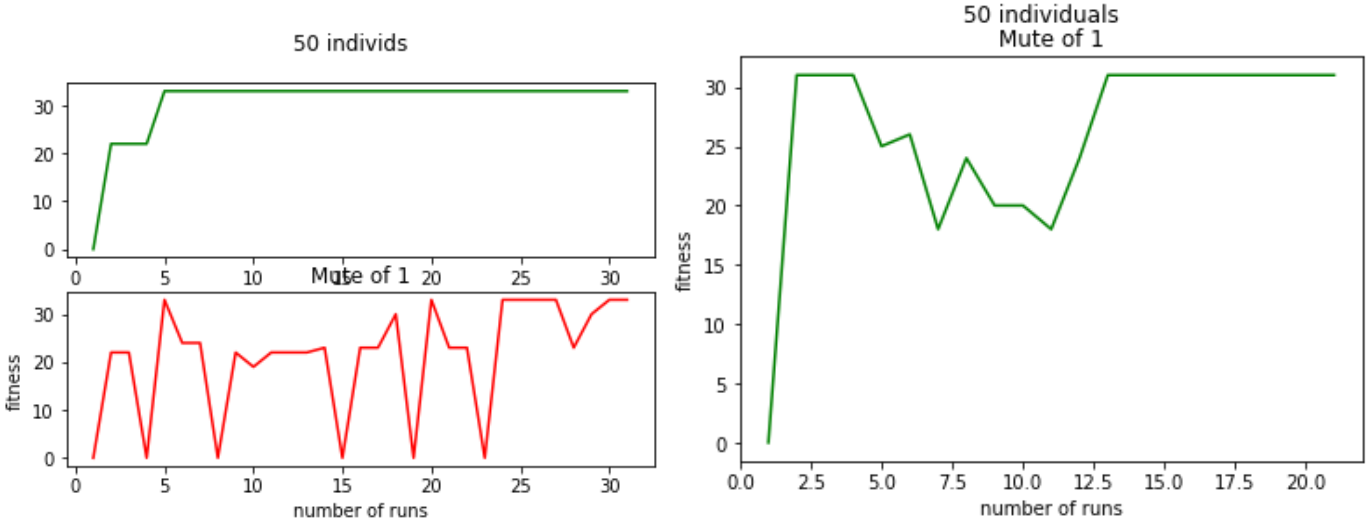


These 2 histograms above prove us that we might not get the best fit at the end of algorithm no matter of how much the parameters change.

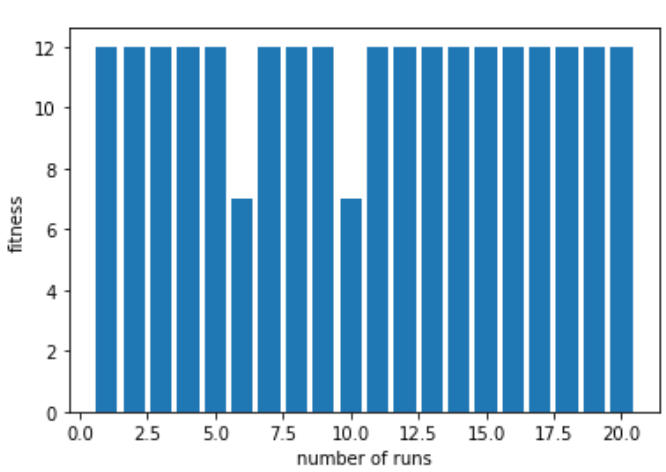


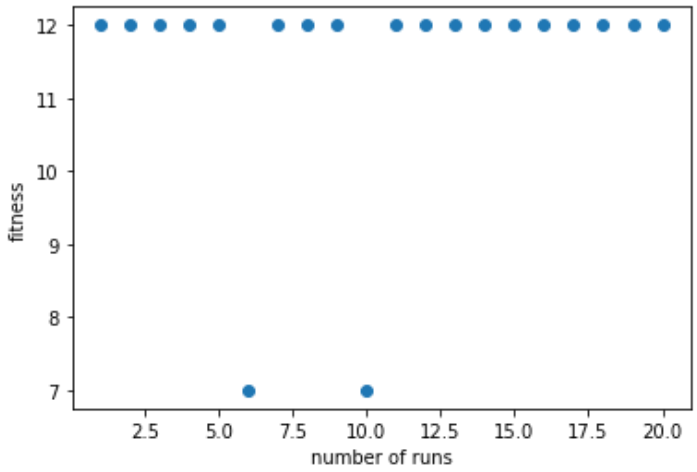
In this example here I showed the performance of 50 random individuals during 10 generations. In this case the great thing about tournament selection is you don't have to pick the individuals proportional to fitness. Also, I am getting rid of population coping using crossover. The population quickly becomes fairly genetically converged. I can produce just one new offspring at a time, replacing one. (Repeat n times for the equivalent of one generation)

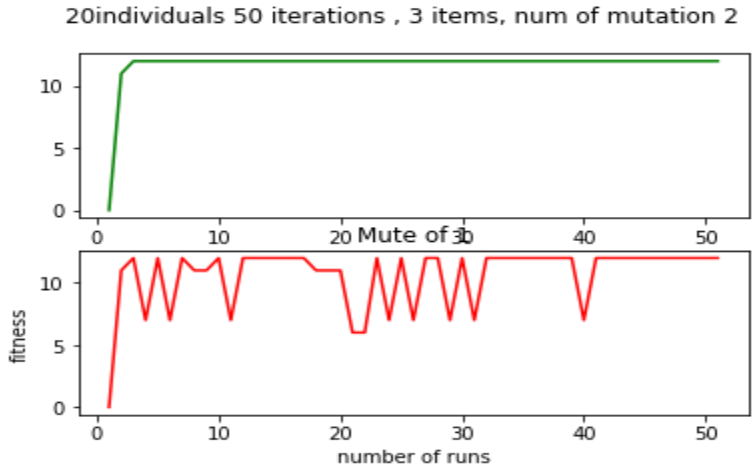
1. Initialise random pop P
2. Evaluate entire pop
3. Pick 2 individuals at random & evaluate them. We call the fitter individual parent one W1 and less fit L1
4. Pick 2 individuals at random & evaluate them. We call the fitter individual parent two W2 and less fit L1
5. Crossover W1 and W2 to produce W1' and W2'
6. Write over L1 and L2 with W1' and W2'
7. Mutate L1 and L2
8. Evaluate new fitness of L1 and L2.
9. Until success or give up, go to 3

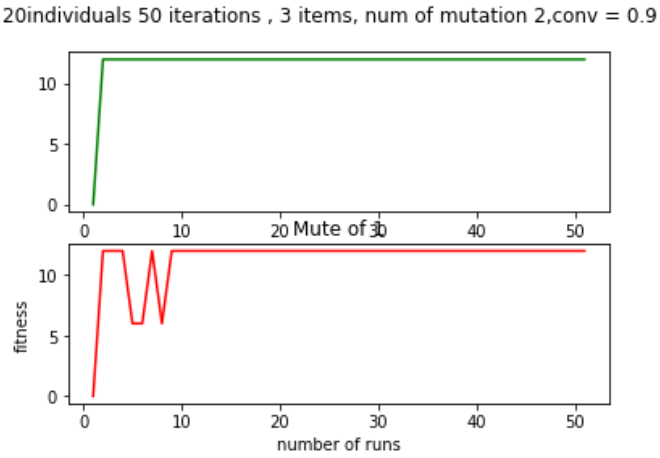


Changing the tournament size and crossover probability over the same 50 individuals in the plot (fitness X number of runs) we spot more variation. Because each new generation is way more different to the previous one it let us explore more possibilities and try different finesses.

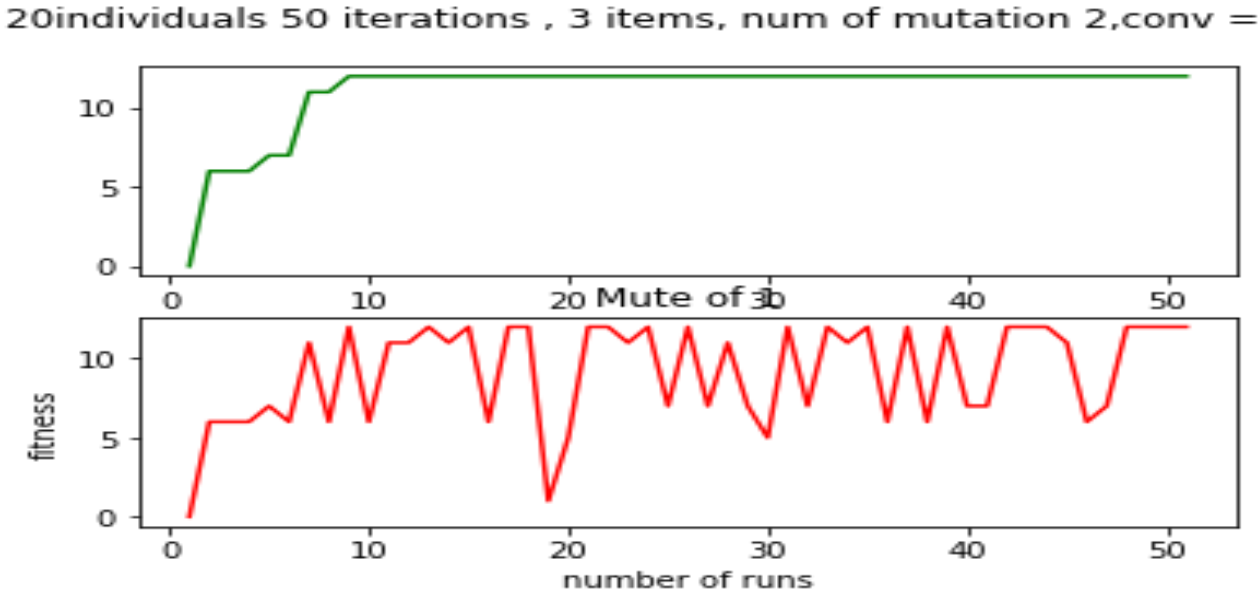








Encouraging diversity, we optimize our GA algorithm and by having a higher mutation rate and crossover probability. In the first plot the mutation rate is one and crossover = 0.5. We see that if we keep more information from parents we reach faster to the desired fitness and vice versa.



Conclusion

Genetic algorithms are evolutionary algorithms which help us solve nonpolynomial mathematical problems. In this report I show how a GA algorithm perform on an NP type problem, Knapsack. The issue with NP type problems is that their time complexity grows veery fast. It will take 5 billion years for my computer to calculate combinations of 71 objects which I want to take on the trip, in my bag pack. Evolutionary algorithms, such a GA, optimize the time complexity and gives us an answer in around a second. The only matter is that at the end of algorithm we might not get the best solution, but for sure we get a solution which satisfies our requirements.

In terms of optimisation, we can do more. I showed above how optimisation works on the performance of an GA and how to optimise the algorithm. By adding a mutation rate, crossover probability we make our offspring unique, as they will never be a product of the first generation. By diversification of the genome, we see improvement in our fitness vs. number of runs. Also, I changed the tournament size and other parameters of the GA to see how algorithm performs in different instances.

As a conclusion, an GA will always find a suited solution, by diversification of the genes (adding a bigger mutation rate and a crossover probability) our algorithm outperforms the previous one.

References

- An Overview of Genetic Algorithms | Emerald Insight. “An Overview of Genetic Algorithms | Emerald Insight.” www.emerald.com, June 1, 1992. https://www.emerald.com/insight/content/doi/10.1108/eb005943/full/html.
- me, Tweet. “Creating a Genetic Algorithm for Beginners.” Creating a genetic algorithm for beginners. www.theprojectspot.com. Accessed April 5, 2022. https://www.theprojectspot.com/tutorial-post/creating-a-genetic-algorithm-for-beginners/3.
- YouTube. “What Is a Genetic Algorithm.” www.youtube.com, September 23, 2015. https://www.youtube.com/watch?v=1i8muvzZkPw&t=1s.
- YouTube. “Genetic Algorithms Explained By Example.” www.youtube.com, July 13, 2020. https://www.youtube.com/watch?v=uQj5UNhCPuo&t=436s.
- YouTube. “13. Learning: Genetic Algorithms.” www.youtube.com, January 10, 2014. https://www.youtube.com/watch?v=kHyNqSnzP8Y&t=23s.