UDC 004.8-043.86

GENETIC ALGORITHMS: HOW COMPUTERS "EVOLVE"

Astrowski V.A.

Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Likhtarovitch I. I. – Senior Lecturer at the Department of Foreign Languages

Annotation. This article discusses the principles of genetic algorithms and the types of problems they are designed to solve. It describes their operating mechanisms, including selection, crossover and mutation, and analyses their advantages over traditional mathematical methods in solving complex optimisation tasks. A practical implementation of the OneMax problem is presented to illustrate the algorithm's functionality.

Keywords: genetic algorithm, selection, crossover, mutation, genes.

***Introduction*.** In 1859, Charles Darwin published his famous work, "On the Origin of Species", in which he articulated the idea of the change of all life on Earth through natural selection, or rather, evolution. However, since biological evolution is believed to take millions, even hundreds of millions, of years, we cannot visualize its action. However, modern computational methods allow us to simulate certain aspects of evolutionary processes.

In 1954, the Italian mathematician Nils Barricelli created a simulation of evolution on a computer at the Institute for Advanced Study, Princeton University. His work provided a powerful impetus for research on this topic, which ultimately resulted in the development of an evolutionary algorithm called the Genetic Algorithm.

A genetic algorithm is a heuristic search method used to solve optimization and modeling problems by applying principles similar to those of evolution in nature. It is based on the fundamental mechanisms of heredity, variation and natural selection.

Main part. A genetic algorithm operates according to the principles of selection, crossover and mutation. To solve a problem using this approach, an initial population of individuals is generated. Each individual represents a potential solution encoded as a chromosome composed of genes. The initial population is typically created randomly to promote genetic diversity. To evaluate the quality of each individual's solution, we must define a fitness function, often expressed as fitness (individual). This function indicates how close a particular individual is to solving the problem. The evolutionary process can then be simulated.

The selection stage follows: its goal is to retain the fittest individuals in the population, while also maintaining population diversity, since the optimal solution can also be achieved by crossing highly fit individuals with less fit ones.

After selection, the algorithm proceeds to the crossover stage, also known in scientific literature as crossing-over. At this stage, data (genes) are exchanged between the chromosomes (solutions) of the parents selected in the first stage. Specifically, two individuals (parents) are selected and chromosome fragments (a group of genes) are swapped, resulting in a new set of genes in the chromosomes of the offspring. This operation is performed with a certain high, but not 100%, probability. Parent individuals that produce offspring may or may not remain in the next generation depending on the algorithm design. With this crossover mechanism, the population size in each subsequent generation will not differ from the initial size.

The final stage in the evolutionary process is mutation. It is applied to the resulting population and randomly changes the values of individual genes or sets of genes with a small probability. The probability of mutation and its strength – that is, how many genes it will replace – are determined by the designer. Mutation is necessary to maintain population diversity and accelerate the search for the desired solution. Although mutation may sometimes produce less optimal solutions, such individuals are typically eliminated during the subsequent selection phase. However, the mutation probability must be carefully chosen, because if it is too large, this may lead to the loss of beneficial genes and not be fixed in subsequent solutions, which will significantly slow down the process of finding a solution.

With a successful mutation, the individual has every chance of passing through selection and producing offspring, which will consolidate the previously obtained good gene in the population.

To illustrate the algorithm, a simple implementation of the OneMax problem was developed in C++. This problem is conceptually simple and clearly demonstrates the functioning of a genetic algorithm. The OneMax problem: we have a sequence of digits of a certain length, which can consist of zeros and ones. Our goal is to find a solution that maximizes the sum of the digits in this sequence. Obviously, the correct solution to this problem is a sequence where each digit in the sequence is one, but this problem provides an easy way to understand how a genetic algorithm works.

The first step involves defining the fitness function. In this case, it is simply a formula 1

$$\text{fitness} = \sum_{i=0}^{N-1} \text{individ}[i] \qquad (1)$$

for the sum of all elements of a numerical sequence. The larger this sum, the more fit the individual is, i.e., the closer to the solution. Next, we create an initial population of these sequences of a certain length, randomly generated. In this example, the population consists of 50 individuals with a long numerical sequence of 80 elements. In this example, an individual (solution) is a single numerical sequence, a gene is a single element of this sequence (0 or 1), and a chromosome is the set of all genes of a given individual.

The next step is selection. For our problem, we will implement tournament selection. The principle of tournament selection is as follows: first, several individuals (in our case, 4) are randomly selected from the entire population, and then the best one – the fittest – is selected. This process is repeated until the number of potential "parents" matches the size of the original population.

After selection, the crossover process is performed. Pairs of parents are chosen, and fragments of their chromosomes are exchanged to create new offspring (Figure 1). This recombination allows advantageous traits from different parents to combine, increasing the likelihood of improved solutions in the next generation.

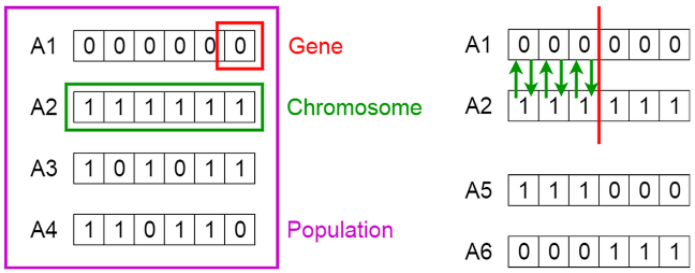

Figure 1 – The crossing principle in the OneMax problem

The final step is mutation. In this study, mutation replaces one gene in an individual's chromosome with a low probability (1%), meaning that a value of 1 may change to 0 or vice versa (Figure 2). This preserves population diversity, which will speed up the solution process.

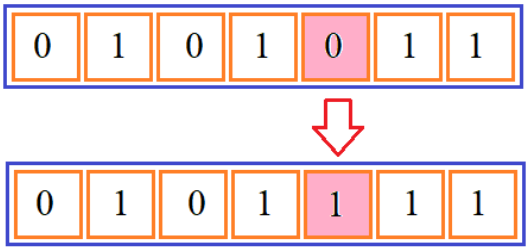

Figure 2 – The mutation principle

The results of the program are shown in Figure 3.

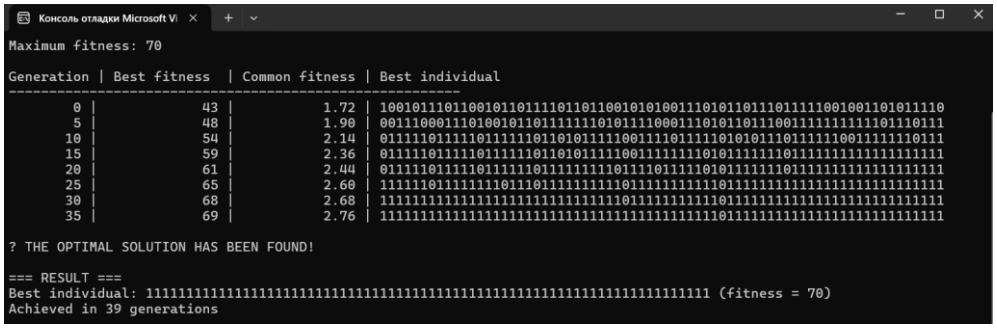

Figure 3 – The program results

All of the above principles were implemented in this program. As we can see, the correct result (the fittest individual) was achieved in 39 generations.

Genetic algorithms are typically used to solve problems that are impossible or very difficult to solve mathematically using formulas and relationships. For example, a genetic algorithm can be used to optimize mathematical functions, create and optimise database structures, create schedules, and even train neural networks.

A well-known real-world example is the use of genetic algorithms in game development. In the 1990s, developers used them to teach virtual creatures how to walk. Each creature had a set of movement parameters encoded as genes, and over many generations the algorithm found combinations that produced efficient and realistic motion. This showed that genetic algorithms can solve problems where the solution is hard to describe in advance but easy to evaluate once found.

***Conclusion*.** This study demonstrates that genetic algorithms represent a powerful optimisation and search tool based on the principles of biological evolution. The analysis and software implementation of the OneMax problem clearly demonstrate how selection, crossover and mutation mechanisms enable consistent improvement of a population of solutions, finding the optimal result even in a vast search space. Thus, evolution, as a fundamental mechanism of biological development, is successfully transferred to the digital environment, opening up broad prospects for solving complex engineering and scientific problems where classical mathematical methods are ineffective.

It is also worth noting that genetic algorithms are not limited to a single implementation strategy. Depending on the nature of the problem, researchers may adapt various parameters such as population size, selection pressure, crossover rate, and mutation probability to achieve better convergence. For instance, in multi-objective optimisation tasks, variants such as NSGA-II (Non-dominated Sorting Genetic Algorithm II) have been developed to simultaneously optimise several competing objectives. These advanced implementations extend the classical genetic algorithm framework and allow it to be applied in fields such as robotics, bioinformatics, logistics, and financial modelling.

One of the key advantages of genetic algorithms over traditional methods is their ability to work without knowing the exact mathematical structure of the problem. A classic optimisation method usually requires derivatives or a clear formula, while a genetic algorithm only needs a way to evaluate how good a solution is. This makes genetic algorithms especially useful in real-world tasks where the problem is too complex or irregular to be described by simple equations. For this reason, they remain a popular and practical tool in modern computer science and engineering. Their continued development and wide adoption across many fields confirm that evolutionary computation is not just a theoretical concept, but a reliable and effective approach to solving the challenges of today.

References

1. Genetic algorithm / Wikipedia., 25.05.2025. – Mode of access : https://en.wikipedia.org/wiki/Genetic_algorithm / Date of access : 25.02.2026.

2. История возникновения генетических алгоритмов [Electronic resource] 26.03.2026. – Mode of access: https://cyberleninka.ru/article/n/istoriya-vozniknoveniya-geneticheskih-algoritmov – Date of access : 25.02.2026.