algorithms

MDPI

Article

Application of Genetic Algorithms for Periodicity Recognition and Finite Sequences Sorting

Mukhtar Zhassuzak [1,2,†], Marat Akhmet [3,†], Yedilkhan Amirgaliyev [1,†] and Zholdas Buribayev [1,2,*]

[1] Department of Computer Science, Al-Farabi Kazakh National University, 71 al-Farabi Ave., Almaty 050040, Kazakhstan; zhassuzak.mukhtar@gmail.com (M.Z.); amir_ed@mail.ru (Y.A.)
[2] LTD DigitAlem, 71 al-Farabi Ave., Almaty 050040, Kazakhstan
[3] Department of Mathematics, Middle East Technical University, Dumlupınar Bulvarı No:1, Ankara 06800, Turkey; marat@metu.edu.tr
* Correspondence: zholdas.buribayev@kaznu.edu.kz
† These authors contributed equally to this work.

Abstract: Unpredictable strings are sequences of data with complex and erratic behavior, which makes them an object of interest in various scientific fields. Unpredictable strings related to chaos theory was investigated using a genetic algorithm. This paper presents a new genetic algorithm for converting large binary sequences into their periodic form. The MakePeriod method is also presented, which is aimed at optimizing the search for such periodic sequences, which significantly reduces the number of generations to achieve the result of the problem under consideration. The analysis of the deviation of a nonperiodic sequence from its considered periodic transformation was carried out, and methods of crossover and mutation were investigated. The proposed algorithm and its associated conclusions can be applied to processing large sequences and different values of the period, and also emphasize the importance of choosing the right methods of crossover and mutation when applying genetic algorithms to this task.

Keywords: unpredictable strings; unpredictable sequence; genetic algorithm; chaos theory; periodic sequences; logistic mapping

Citation: Zhassuzak, M.; Akhmet, M.; Amirgaliyev, Y.; Buribayev, Z. Application of Genetic Algorithms for Periodicity Recognition and Finite Sequences Sorting. *Algorithms* **2024**, *17*, 101. https://doi.org/10.3390/a17030101

Academic Editors: Nikola Ivković and Matej Črepinšek

Received: 18 January 2024
Revised: 10 February 2024
Accepted: 17 February 2024
Published: 26 February 2024

1. Introduction

Unpredictable strings, often associated with chaos theory, represent data sequences that exhibit complex and disorderly behavior, making them an intriguing subject of study in various scientific disciplines. In chaos theory, unpredictable strings are characterized by their sensitivity to initial conditions, where even small changes in the initial values can lead to entirely different outcomes, emphasizing the nonlinear nature of chaotic systems [1].

Research on unpredictable strings and randomly determined unpredictable functions has significant potential advantages in various domains today. Specifically, this research field plays a significant role in cryptography and information security for generating reliable encryption keys, random initialization vectors, and secure random numbers. It also plays a crucial role in ensuring the confidentiality and integrity of communication channels, which can enhance communication protocols protecting confidential data from interception and tampering.

For example, in [2], a method for synchronizing two fast random bit generators (RBG), based on coupled chaotic lasers, was developed. This research addresses the issue of ensuring security in synchronizing RBGs in fast physical systems, where predictability is eliminated. Furthermore, the study of unpredictable strings has introduced a new concept of unpredictable sequences of a finite number of symbols [3]. The first and second laws of large sequences for random processes in discrete time were proven, with the latter being closely related to Bernoulli's theorem.

If we consider an infinite sequence a_i, then an unpredictable string of length k is defined as a finite array $a_s, a_{s+i}, \ldots a_{s+k}$, where s and k are positive integers, subject to the following conditions: $a_i = a_{s+i}$ for $k = 0, 1, 2, \ldots, k-1$, and $a_i \neq a_{s+i}$. In other words, we consider a section of a sequence of length k, starting from position s, where the values are repeated in increments of k, but the next element after this section differs from the element standing at the position of the first element of the repeating sequence. This definition allows you to identify a part of the sequence with predictable repetition and the unpredictable part following it. These unpredictable strings play an important role in defining unpredictable sequences with a limited number of characters. The sequence a_i itself is considered unpredictable if it allows the existence of unpredictable strings of arbitrary length.

This definition describes an array with a repeating sequence of length k, starting at position s, and it is considered unpredictable if the element following the repeated sequence a_k is different from the element at the same position as the first element of the sequence a_{s+k}. This means that the array has a predictable part followed by an unpredictable part. Unpredictable strings are used to define unpredictable sequences of a finite number of symbols. A sequence a_i is unpredictable if it allows unpredictable strings of arbitrarily large length. Stochastic processes with discrete time and finite state spaces allow for a countable set of realizations that are unpredictable sequences [3].

Researching this topic will allow for a better understanding of fundamental concepts related to chaos in statistics, random processes, probability theory, and dynamic processes [4].

In the paper [5], the application and evaluation of a cryptosystem based on chaotic mappings are investigated. An analysis was conducted to assess the impact of the chosen chaotic mapping on the properties of messages and the security of the cryptosystem using methods from statistical mechanics and information theory.

The paper [6] explores a synchronization scheme for complex networks of chaotic systems. Chaotic systems, including Rössler, Chen, Lorenz, and Lü, are considered as complex chaotic systems within complex networks. This work specifically focuses on the application of the control law obtained for synchronizing an irregular network consisting of six different chaotic systems. The paper underscores the utility and advantages of the proposed synchronization scheme through numerical simulations of complex chaotic networks.

In another work [7], the possibility of creating a secure optoelectronic communication system using chaotic Rössler oscillators and semiconductor lasers is investigated. The results confirm that the system is viable.

In the paper [3], a numerical analysis of the Bernoulli process with periodic realizations is carried out. Overall, the study of periodic sequences allows for a deeper understanding and analysis of the structure and characteristics of unpredictable sequences, as well as the identification of regular patterns and other key aspects [8].

This paper represents the first step in exploring unpredictability through deep learning methods. The research focuses on the periodicity of random symbolic sequences, which is one of the elements of chaos [9]. Therefore, the investigation of this problem serves as a prerequisite for further delving into the subject matter. In this paper, the object of the study was a binary sequence.

The study of periodic sequence is due to a wide range of applications in science and practice. For example, in signal processing, periodic sequences are important. The study of their properties will allow you to gain a good understanding in the generation and control of signals, as well as in solving problems related to filtering and signal processing. It can also be useful in the field of data encoding and decoding. They can be used as part of a code system that monitors the need to transmit information [10].

The advantages and disadvantages of the aforementioned papers in this section are outlined in the following Table 1.

Table 1. Advantages and disadvantages of the used papers.

Title	Features	Advantages	Disadvantages
[1]	The complex and disorderly behavior of unpredictable strings in the context of chaos theory.	Reflection of sensitivity to initial conditions.	The nonlinear nature of chaotic systems.
[2]	Synchronization of physical random bit generators based on chaotic lasers and their ç application.	Solving the security problem of RBG synchronization in fast physical systems. Generation of unpredictable bit strings for cryptography and stochastic simulations.	It requires complex technologies and equipment, which can lead to difficulties in implementation in real conditions.
[3]	The introduction of a new concept of unpredictable strings and the definition of deterministic unpredictable sequences on a finite number of characters. Numerical analysis of the Bernoulli process with periodic realizations.	The development of a new concept of unpredictable sequences. Proof of the first law of large strings for random processes in discrete time.	A limit on the length of sequences.
[4]	An introduction to probability theory and stochastic processes with applications.	Providing a clear and understandable approach to probability theory and stochastic processes.	No significant deficiencies have been identified.
[5]	Implementation and evaluation of a block cryptosystem based on chaotic mappings using high-precision approximation methods.	The use of chaotic mappings to ensure security in block cryptosystems.	There may be difficulties in assessing the randomness of cryptograms.
[6]	Synchronization of complex networks of chaotic systems through control corresponding to the model.	The application of a synchronization scheme for complex networks from chaotic systems of Rössler, Chen, Lorenz, and Lü. Special attention is paid to the application of the governing law to synchronize an irregular network of six different chaotic systems. Emphasizes the usefulness and advantages of the proposed synchronization scheme by conducting numerical simulations of chaotic complex networks.	Possible difficulties in real-world implementation.
[7]	Creation of a secure optical communication system using Rössler chaotic oscillators and semiconductor lasers.	Investigation of the possibility of secure optical communication using chaotic oscillators and lasers.	Limitation on specific applications in real-world scenarios.

Table 1. *Cont.*

Title	Features	Advantages	Disadvantages
[8]	It opens up opportunities for deep understanding and analysis of random character sequences in the context of chaos.	The development of a randomly defined unpredictable function. Analysis of the Bernoulli process with periodic implementations. The possibility of numerical analysis and investigation of periodic sequences. Simulation of a randomly defined unpredictable function.	The text mentions conducting numerical simulations, but there is no detailed description of the experiments themselves.
[9]	The study of unpredictability by deep learning methods.	An extensive consideration of chaos in dynamical systems. An excellent source for understanding and analyzing chaotic dynamical systems. Application of chaotic systems concepts in various fields of science and practice.	Possible difficulties for beginners in the topic of chaos.
[10]	Search for hidden periodicity in amino acid sequences using a genetic algorithm and dynamic programming.	The study of amino acid sequences in order to find hidden periodicity. The use of a genetic algorithm and dynamic programming to search for hidden periodicity. Determining the importance of periodicity in amino acid sequences. The research has potential applications in the field of genetics and molecular biology.	Possible limitations in the field of generalization of results.

2. Related Work and the Current Contribution

The main goal of this paper is to investigate and develop an efficient method for transforming binary nonperiodic sequences into their periodic analogs using a genetic algorithm. The research aims to enhance algorithms, especially in the context of their dependence on unpredictable input data. As a specific contribution, the paper introduces the concept of error for nonperiodic sequences, develops an efficient genetic algorithm, and proposes a new method called "MakePeriod", significantly improving the algorithm's convergence. The described approach is an innovative method that combines aspects of unpredictability and artificial intelligence. The use of logistic mapping to construct the test sequence adds a dynamic element to the research, opening new possibilities in applying artificial intelligence methods for analyzing periodicity in chaotic systems.

2.1. Summary of Contributions

1. Introduction of the error concept for nonperiodic sequences: A new approach to evaluating the transformation of nonperiodic sequences into periodic ones by defining an error measuring the differences between them.
2. Development of an efficient genetic algorithm: The paper presents a genetic algorithm for transforming nonperiodic sequences using a heuristic approach and a state enumeration method for optimization.

3. Introduction of the "MakePeriod" method: The paper introduces the "MakePeriod" method, significantly reducing the number of generations needed to obtain a solution, significantly contributing to the algorithm's efficiency.
4. Analysis of crossover and mutation methods: Comparative analysis of various crossover and mutation methods, highlighting optimal combinations for achieving the best results in the specific task.
5. Providing perspectives for application in large sequences: The research emphasizes the prospects of applying the proposed method for processing large sequences and period values.

2.2. Applications

The paper's contribution lies in proposing a new method for efficiently transforming nonperiodic sequences into their periodic analogs using a genetic algorithm, which can have wide applications in various fields such as data approximation and algorithm optimization. More specifically,

1. **Chaos and Dynamic Systems Research:** Understanding and analyzing chaotic systems using periodicity processing methods.
2. **Bioinformatics:** Analyzing genetic sequences where periodicity may be related to specific genetic patterns or structures. Investigating periodicity in biological data, such as amino acid sequences, to identify patterns.
3. **Cryptography and Information Security:** Generating secure encryption keys, random initialization vectors, and secure random numbers for cryptographic applications.
4. **Signal Processing and Telecommunications:** Analyzing periodicity in signals to improve filtering, modulation, and decoding processes in telecommunication systems.
5. **Medical Diagnostics:** Researching periodicity in biomedical data to detect regular patterns associated with diseases or physiological processes.
6. **Financial Analytics:** Analyzing time series of financial data to identify periodic trends and patterns in the market.

3. Problem Statement

Studying unpredictable strings and functions will enable the improvement of specific algorithms in terms of enhancing their efficiency because many of these types of algorithms heavily rely on the unpredictability of input data. Researching the role of unpredictability allows for the optimization of these algorithms and the enhancement of quality assurance. This is particularly significant as the quality and unpredictability of generated sequences hold key importance in many applications [11].

Our task is to obtain all possible periodicity options for a given sequence. A new genetic algorithm is used to solve the problem. In turn, the development of this genetic algorithm consists in finding optimal solutions in a finite set of objects that will meet certain criteria.

A logistic mapping is used to construct the sequence under test. This is a discrete mapping that is often used to model dynamic systems with limited resources or to generate binary sequences [12]. Despite the simplicity of the logistic mapping, it allows you to model very complex processes. Our choice to use the logistic mapping is justified by the desire to take advantage of the simplicity of the formula, but we also emphasize that the logistic mapping is, in fact, a special case of a more general class of chaotic maps. Chaotic mappings, including logistic mappings, provide important tools for creating dynamic and complex sequences with interesting characteristics. Although we used a specific logistic mapping to create a sequence, it is also important in our study to further consider more general concepts of chaos. The logistic equation has the form

$$x_{n+1} = \mu x_n(1 - x_n), \quad x \in \mathbb{R}, \tag{1}$$

where x_n is the current value, and x_{n+1} is the next value. The parameter μ controls the behavior of the system.

The choice of the parameter μ and the initial value x_0 in the equation strongly affect the generation of the binary sequence. It is convenient to take the parameter μ in the interval [0, 4] [13]. Two areas, F_1 and F_2, are also selected, which, depending on its value, will relate to x. An element of a binary sequence will have the value 0 or 1 if it belongs to the domains F_1 and F_2, respectively:

$$s_i = \begin{cases} 0, & x_i \in F_1 \\ 1, & x_i \in F_2 \end{cases} \quad (2)$$

Thus, to construct a binary sequence, the value of the parameter μ = 3.91, and the initial value x_0 = 0.4. Since we needed to obtain the sequence s_n, which will consist of 0 and 1, the regions were chosen as follows, respectively: F_1 = [0, 0.5] and F_2 = [0.5, 1].

For a computational program, a sequence consisting of 0 and 1 is considered as input data. A genetic algorithm is applied to it, which solves the problem by methodically iterating through states and state transitions in order to find a solution from the initial state to the final one. The task of the algorithm is to make a calculation based on a heuristic approach, which, for the initial length of the considered (input) sequence s_n, checks it for periodicity P, after which it outputs all its received variants converted into a periodic form. The process of checking the sequence s_n for periodicity P occurs by comparing $s_i = s_{i+P}$ the value of each element with $i \leq N - P$, where N is the length of the sequence.

Heuristic algorithms are widely used to solve problems of high computational complexity. The idea is that instead of a complete search of options, which takes a time-consuming process, a relatively easy approach is used. The main disadvantage of this approach is an insufficiently theoretically sound algorithm [14].

4. Genetic Algorithm

Genetic algorithms belong to a family of search algorithms whose ideas are suggested by the principles of evolution in nature (Figure 1). The main task of such algorithms is to simulate the processes of natural selection and find high-quality solutions to problems. At the same time, the analogy with natural selection allows these algorithms to overcome some obstacles that stand in the way of traditional search and optimization algorithms, especially in problems with big data. Just as Darwinian evolution promotes the development of individual organisms in a population, genetic algorithms contribute to the development of a population of potential solutions to this problem, which are called "individuals". These solutions are periodically evaluated and used to form a new generation of solutions. Those individuals who have shown the best results in solving the problem are more likely to be selected and pass on their positive characteristics to the next generation. Thus, gradually potential solutions become more perfect in solving this problem [15].

In the natural environment, crossover, reproduction, and mutation are realized with the help of a *genotype*—a set of genes ordered in chromosomes. When two individuals are crossed and offspring arise, each chromosome of the offspring carries a mixture of genes from both parents. In the course of its work, a genetic algorithm always contains a set of individual solutions, which is called a population. Each of these individuals is represented by its own chromosome, which allows us to consider the population as a collection of chromosomes. For a software implementation, of course, this analogue will be like a set of arrays that store certain bit values created randomly at the beginning.

At each step of the algorithm execution, individual solutions are evaluated through the *objective function*. This function evaluates the quality of the solution of the considered problem of the genetic algorithm. All individuals in a generation receive their personal assessments from the fitness function. The most suitable individuals with a higher probability of their decision will be *selected* for reproduction and integration into the next generation. However, this does not deny that individuals with low fitness will not be selected. This allows their genetic material not to disappear, but their probability of selection will be low.

After that, a *crossover operation* takes place, which simulates biological crossover with the preservation of the genes of the parents' heredity. This operation is used to combine the genetic information of two individual parents in the process of creating offspring. Crossover is not always used, but is applied with a certain probability. If the crossover is not performed, the genetic images of both parents are transmitted to the next generation without changes.

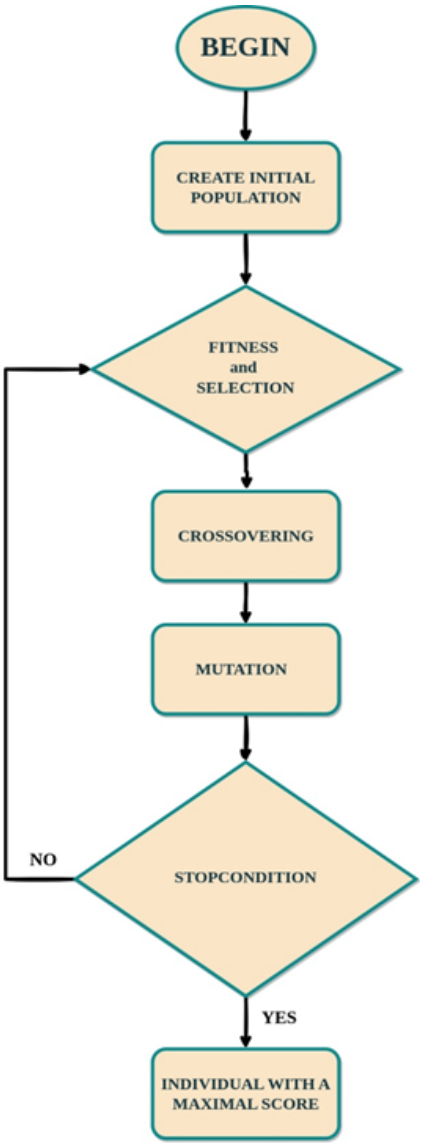

Figure 1. The scheme of the genetic algorithm.

In the end, a *mutation operator* is executed, the task of which is to periodically randomly update information in the population. This will allow the introduction of new combinations of genes into individuals, which in turn will allow the exploration of unexplored areas of the solution space. A mutation can manifest itself as a random change in a single gene [16].

4.1. Genotype and Population

In this problem, we will consider the genotype as a binary string in which each element corresponds to one of the characters 0 and 1. Then, for the problem under consideration, the population will have the form.

As can be seen from the Figure 2, an area of sequences is considered as input data, the initial values of which will be randomly generated.

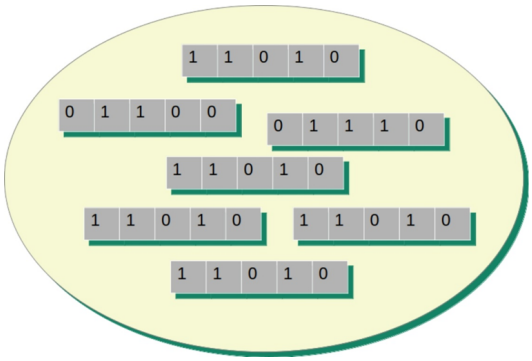

Figure 2. Illustration of genotypes in a population for the problem under consideration.

In the genotype of each individual of the population under consideration, a digital value of 0 or 1 is encoded. That is, we represent periodic structures in the form of a sequence of bits in the genotype. A population includes a set of such genotypes, where each genotype represents a potential solution to the problem. The genetic algorithm is focused on solving the problem of finding the periodicity in a binary sequence. The input data are a binary sequence of 0 and 1. The purpose of the genetic algorithm is to solve the problem of methodically iterating through states and transitions between states in order to find a solution from the initial state to the final one. The genetic algorithm uses a heuristic approach to effectively check the frequency of the input sequence. The verification process is based on comparing the values of each element of the sequence with the previous elements. The frequency checking algorithm compares the value of each element with the previous values for a possible period. This process helps to identify periodic sequences. The found periodic sequences are presented in the form of various variants in a periodic form. The search space is determined by the length of the input sequence, and the genetic algorithm goes through various combinations and state transitions in order to find periodic structures. The assessment of the fitness function includes a periodicity check to ensure that the sequence does not consist only of the values 0 and 1. This is done to prevent monotonous solutions with low adaptability. Defining the feature search space: The feature search space is defined through the genotype configuration.

4.2. Fitness Function

To evaluate the fitness function at each iteration for the problem under consideration, it was necessary to make a *periodicity* check so that the sequence *did not consist of only the value* 0 or 1.

The work of the function to determine the periodicity of each individual has the following pseudocode (Algorithm 1):

Algorithm 1 Function CheckPeriodOrNo(individual, Period)

```
for index ← 0 to (N − Period) do
    if individual[index] = individual[index + Period] then
        checker ← True
    else
        checker ← False
    end if
end for
if checker = True then
    return True
end if
```

It is known that even sequences consisting of a single value of 0 or 1 are periodic. Therefore, we will not consider this solution. Thus, it was necessary to create two additional auxiliary functions: CheckSameOrNo and ExistOrNo. The CheckSameOrNo function just

checks the above, and the second ExistOrNo function checks for the identity of the found solution in the solution area.

$$\texttt{CheckPeriodOrNo}(\text{indiv.}, \text{Period}) = \begin{cases} \text{True}, & \substack{\forall \text{idx} \in [0, N-\text{Period}-1]: \\ \text{indiv.[idx]}=\text{indiv.[idx + Period]},} \\ \text{False}, & \text{otherwise.} \end{cases}$$

$$\texttt{CheckSameOrNo}(\text{indiv.}) = \begin{cases} \text{True}, & \forall \text{idx}_1, \forall \text{idx}_2 \in [0, N-1] : \text{indiv.}[\text{idx}_1] = \text{indiv.}[\text{idx}_2], \\ \text{False}, & \text{otherwise.} \end{cases}$$

$$\texttt{ExistOrNo}(\text{solutions}, \text{indiv.}) = \begin{cases} \text{True}, & \text{indiv.} \in \text{solutions}, \\ \text{False}, & \text{otherwise.} \end{cases}$$

$$\texttt{CalculateFitness}(\text{indiv.}) = \begin{cases} \text{True}, & \texttt{CheckPeriodOrNo}(\text{indiv.}, \text{Period}) = \text{True} \\ & \texttt{CheckSameOrNo}(\text{indiv.}) = \text{False} \\ & \texttt{ExistOrNo}(\text{solutions}, \text{indiv.}) = \text{False}, \\ \text{False}, & \text{otherwise.} \end{cases}$$

Thus, the objective function itself has the following pseudocode (Algorithm 2):

Algorithm 2 Function CalculateFitness(individual)

```
fitness ← 0
if CheckPeriodOrNo(individual, Period) = True then
    if CheckSameOrNo(individual) = False then
        if ExistOrNo(solutions, individual) = False then
            fitness ← fitness + 1
        end if
    end if
end if
return fitness
```

4.3. Crossover and Mutation

The operations of crossover and mutation are the main operators for finding a solution to the problem under consideration. Crossover is necessary to combine the genetic information of two individuals acting as parents in the process of generating offspring. As a rule, the crossover operator is not always used, but with some probability. If crossover is not applied, then copies of both parents pass into the next generation without modification. For our task, we will consider three types of crossover: single-point, two-point or k-point, and uniform.

In the book [17], the single-point crossover operator is considered as one of the most common methods for breeding in binary encoding. Single-point crossover is a method where two parental chromosomes selected from the population are cut at a random point known as the crossover point. The genetic information located to the left (or right) of this point is exchanged between the two parents, forming two new offspring. Figure 3 shows an example of a single-point crossover of two pairs of individuals as parents and the creation of their new offspring:

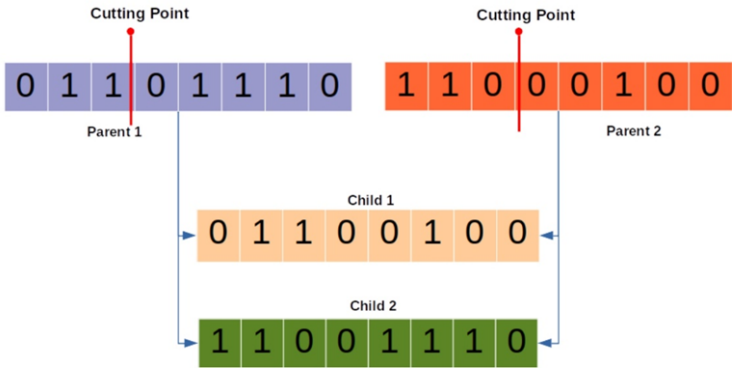

Figure 3. Example of single-point crossover.

In the paper [18], operators of the k-point crossover type are investigated, along with their combinatorial, graphical, and topological properties. The authors demonstrated that k-point crossover operators have a more complex nature and correspond to objects of higher dimensionality, whereas single-point crossover operators correspond to circles, which are relatively simple two-dimensional objects [19]. The principle of operation of a two-point crossover is similar to that of a single-point one, only two or more crossover points in each chromosome are randomly selected. Thus, information from one chromosome located between these points is exchanged by similarly located genes of another chromosome (Figure 4).

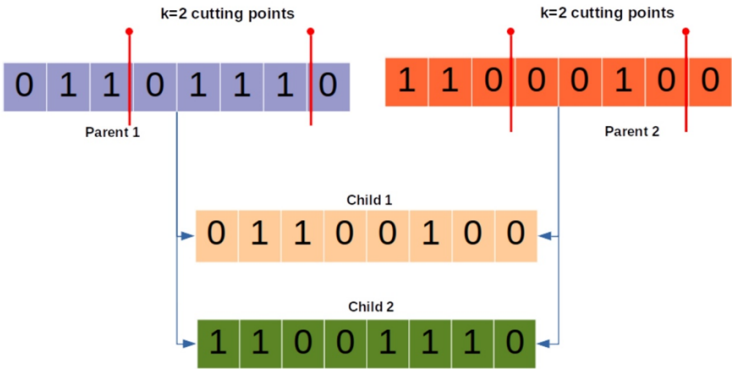

Figure 4. Example of two-point crossover.

In the paper [20], a genetic algorithm is proposed to solve the problem of ordering and scheduling the arrival of aircraft in systems with multiple runways, using uniform crossover. In this paper, the uniform crossover operator ensures efficient identification, transmission, and protection of common traffic subsequences, while preserving the ability to diversify chromosomes. With uniform crossover, each gene in the offspring is selected with the same probability either from the corresponding gene of the first parent or from the second. The bottom line is that for each gene, there is a 50% chance of choosing it from one parent and a 50% chance of choosing it from another parent. This method allows for a more diverse exploration of the solution space compared with other crossover methods, such as single-point or two-point crossover. Uniform crossover can be especially useful when solving a problem may require optimal mixing of different characteristics from both parents. Uniform crossover allows for maintaining genetic diversity in the population [21]. An example of an illustration of the work of uniform crossover is shown in Figure 5:

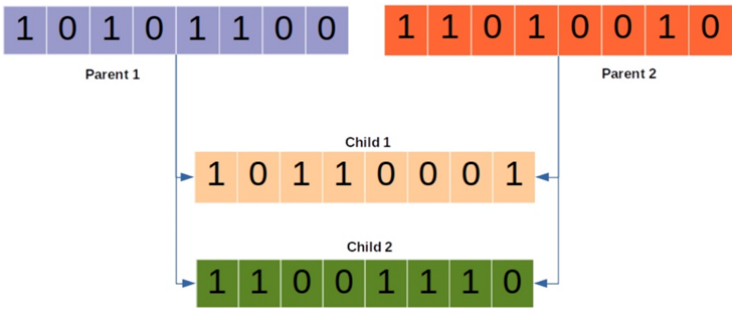

Figure 5. Example of uniform crossover.

Child 1: 10110001 (1 from parent 1, 0 from parent 2, 1 from parent 1, 1 from parent 2, 0 from parent 1, 0 from parent 2, 0 from parent 1, 1 from parent 2).

Child 2: 11001110 (1 from parent 2, 1 from parent 2, 0 from parent 1, 0 from parent 1, 1 from parent 2, 1 from parent 1, 1 from parent 2, 0 from parent 1). Thus, each information from the genes of a descendant is selected from one parent with the same probability. A mutation for the binary chromosome in question consists in changing a random bit or several bits in the chromosome with some probability. Mutation introduces randomness into the genetic algorithm, helping to avoid getting stuck in local optima and ensuring diversity in the population.

Mutation is a genetic operator applied to the offspring resulting from selection and crossover. The mutation operation is probabilistic and is employed to prevent stagnation and ensure diversity in the population. In the problem under consideration, three types of mutation were selected: *bit inversion, mutation by exchange, mutation by reversion*.

Inversion of bits is a mutation method proposed in the original genetic algorithm by Holland [22]. This method is simple and effective for working with binary chromosomes. Inverting a bit means changing the value of the bit to the opposite: if the bit was equal to 0, then after inverting it, it becomes equal to 1, and vice versa (Figure 6). In our task, we will apply as part of the mutation operator for diversity in the population and search for a wider solution space.

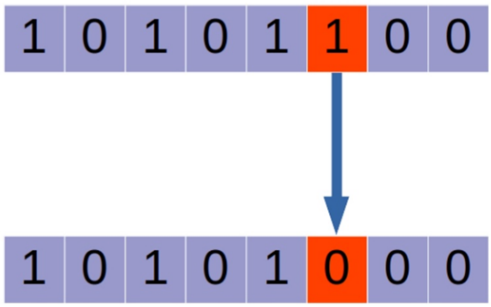

Figure 6. Example of bit inversion.

Swap mutation is a mutation method applicable to both binary and integer chromosomes. It involves randomly selecting two genes in the chromosome and exchanging their values. This mutation operation is suitable for chromosomes representing ordered lists since, after the mutation, the set of genes in the new chromosome remains the same as in the original, except for the random change in the positions of the two specific genes (Figure 7).

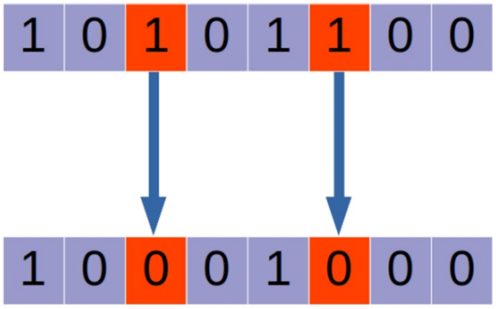

Figure 7. Example of mutation by exchange.

Reverse mutation is a mutation method applied to both binary and integer chromosomes. In this method, a random sequence of genes is selected, and their order within the chromosome is reversed. Similar to the swap mutation, this method is suitable for chromosomes representing ordered lists since, after the mutation, the set of genes in the new chromosome remains the same as in the original, except for the reversal of the order of the selected genes (Figure 8).

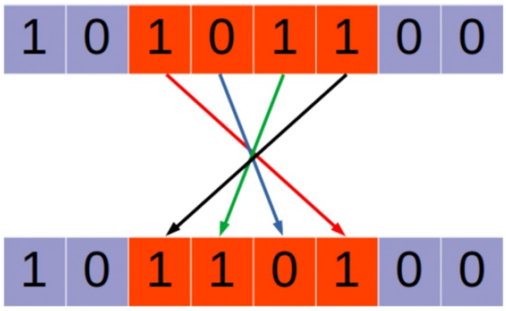

Figure 8. Example of mutation by reversion.

4.4. MakePeriod Method

Finding all the periodicity variations for a larger sequence size did not show a very good result in time. More specifically, for the sequence size $n > 30$, the algorithm did not find solutions for more than 30% of the total and got stuck for a very long time. In this regard, it was decided to optimize the algorithm. Thus, an additional MakePeriod method (Algorithm 3) was created, which performs a modification operation of a binary list in order to provide a certain periodic behavior inside this list. This method is used to create a periodic structure, ensuring that the elements of the list correspond to an alternating pattern in a given period P.

Algorithm 3 Function MakePeriod(sequence, Period)

```
for index ← 0 to (N − Period) do
    if sequence[index] ≠ sequence[index + Period] then
        if sequence[index] = 1 then
            sequence[index + Period] ← 1
        else
            sequence[index + Period] ← 0
        end if
    end if
end for
return sequence
```

As a result of executing this method, the elements of the *sequence* list will be modified in such a way as to provide a given sequence periodic for a certain P. That is, this operation will allow the algorithm to change individuals who are nonperiodic to periodic ones, which significantly reduces the search time for a solution.

Example 1. *Let the sequence be nonperiodic, and we will check it for the value of period 3, that is,* $s_n = 1001010, P = 3$.

Starting from the third element, we see that the periodicity rule for 3 is not fulfilled for this sequence (the values of the elements in indexes 3 and 5 are different). However, we can change the value in index 5 to the opposite one, which has index 3 (Figure 9).

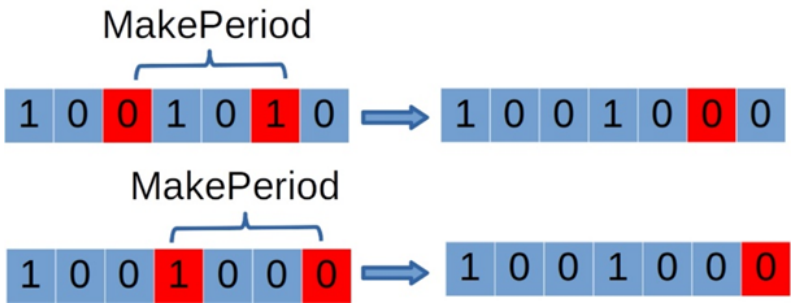

Figure 9. Example of the MakePeriod method.

This will avoid generating additional individuals and their checks, because without the MakePeriod method, our algorithm would simply not consider this sequence, since it is nonperiodic. Thus, a new sequence will be added to the area of potential solutions, which will allow us to narrow the search area of the solution and reduce the time.

The proposed algorithm is iterative; then it is known that a breakpoint must be determined to stop it. We need to find the number of possible sequences that are periodic. Experimentally, for the initial values of the periods, we obtained a pattern that these are two to the power of the period value under consideration (2^p). Given the fact that we will not consider the sequence as a solution that consists of identical elements (such sequences are 2), we have the following formula:

$$k = 2^p - 2 \tag{3}$$

This Formula (4) determines the breakpoint; that is, if the number of solutions found is equal to this value, then the genetic algorithm stops working. It is important to note that determining the specific value of the breakpoint is one of the reasons for further research in this direction. The complete scheme of operation of the algorithm in question is indicated in the form of a block diagram (Figure 10).

As can be seen from the diagram, each individual is checked for periodicity during selection; those that are really periodic are added to the list of solutions, and for those that are not periodic, the MakePeriod operation is performed. When executing the MakePeriod operation for a given sequence, if it meets the periodicity condition and is distinct within the list of solutions, it is also added to the list of solutions. If the above conditions are not met, then the crossover and mutation operations occur sequentially. After that, the stop condition is checked. If the number of solutions is equal to k, which is determined by Formula (1), then the algorithm stops working; if not, a selection operation is performed for already-changed individuals.

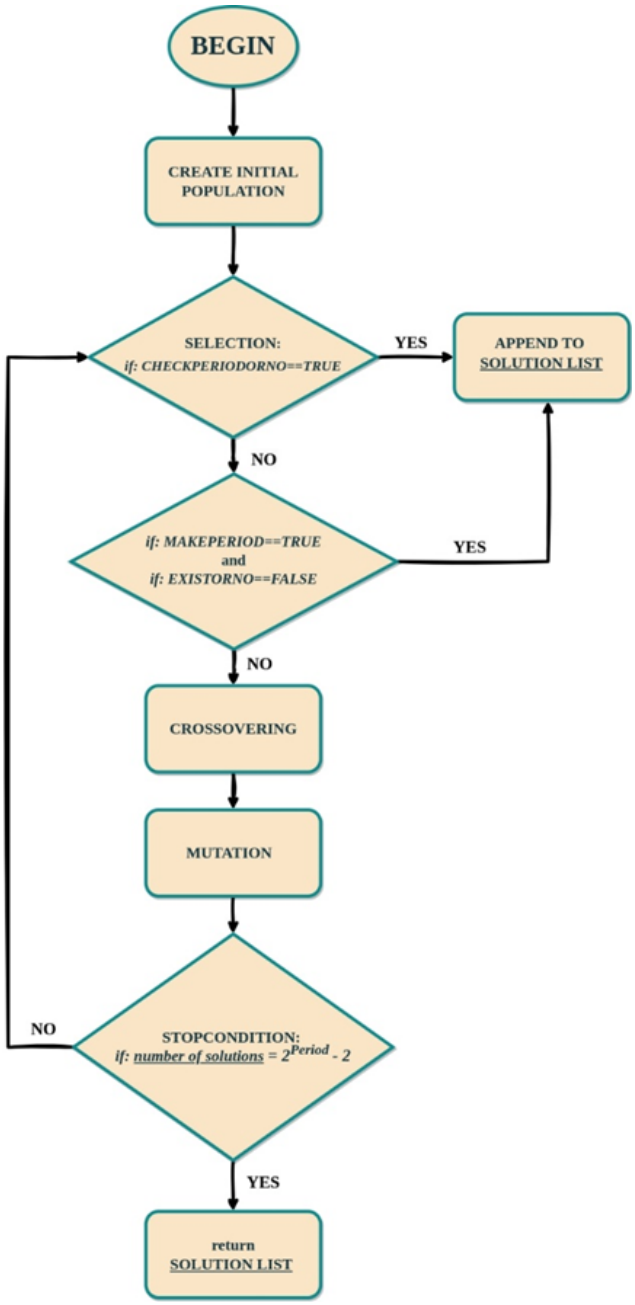

Figure 10. Algorithm operation scheme.

4.5. The Concept of Error for Nonperiodic Sequences

After we have received a certain amount of variation to transform nonperiodic sequences into periodic ones, we need to choose one of them for further research. Thus, the concept of error was defined for such sequences. We will call an error the deviation of a nonperiodic sequence from its considered periodic transformation. The error in converting a nonperiodic sequence into a periodic sequence can be defined as the difference between the original nonperiodic sequence and the best periodic approximation of the same sequence. This error is measured by a value that can be called an "error" between the original and approximated sequences. For each resulting solution (variation), we will compare it with the input periodic sequence and the number of its modified elements:

$$E = \sum_{i=1}^{N} \frac{\delta(a_i, t_i)}{2^i} \tag{4}$$

where

N = the size of the sequence in question

a_i = an element of the input nonperiodic sequence at the position i

t_i = the element of the periodic sequence at the position i

$$\delta(a_i, t_i) = \begin{cases} 1, & \text{if } a_i \text{ is not equal to } t_i \\ 0, & \text{otherwise} \end{cases}$$

This formula is the sum of the values of the function δ for each pair of elements of two sequences, a_i and t_i, which will calculate the number of modified elements between them. In the end, we divide by the value 2^i, since this is a convergent series, i.e., the contribution to the error of those elements that are far away is small. In the beginning, the error values will be relatively significant. After calculating all the error values, the one that best approximates the original nonperiodic sequence to its periodic approximation is selected. Choosing the option with the smallest error is standard practice when optimizing and approximating data. It allows us to find the best solution for the problem of converting a nonperiodic sequence into a periodic form in terms of minimizing the differences between them.

5. Results and Analysis

5.1. Results of the MakePeriod Method

To identify the effectiveness of the result of the algorithm, we first checked without the MakePeriod method (from the diagram in Figure 1) and with its addition on the following simple input data:

The sequence $\mathbf{s}_{15}$, obtained with the help of the logistic map, i.e., with the size $n = 15$ elements, is considered. It was necessary to find all variants of its transformation from this sequence to obtain periodicities at $p = 2, 3, 4$. In the population, the number of individuals was selected to be 50, and the probability of their mutation was 0.5.

$\mathbf{s}_{15}$ = 0 1 0 1 1 1 1 0 0 1 1 1 0 0 1 0

If, for $p = 4$, the algorithm without adding the MakePeriod method found all 14 solutions in 2314 generations, then it coped with the addition in 2 generations:

From the Figure 11, it is evident that the introduced method significantly improves the optimization of the search for the desired solutions.

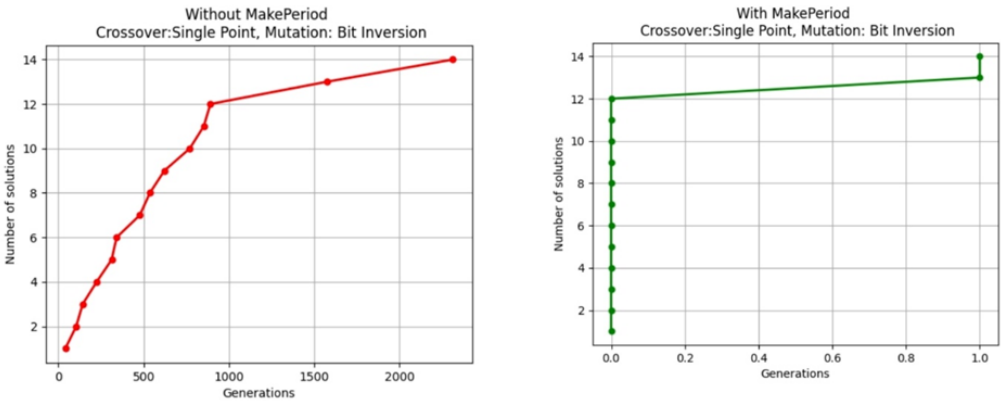

Figure 11. The results of the algorithm without the addition of MakePeriod and with the addition.

5.2. Crossover and Mutation

To evaluate the work of the genetic algorithm, we tested three types of crossover and three types of mutation for each in order to identify their optimality.

First of all, using the logistic map, a binary sequence was also obtained, but already consisting of n = 100,000 elements. This sequence is nonperiodic. Our task was to identify an effective algorithm from the above. Therefore, we decided to calculate all the variants of its transformation in order to obtain a periodicity at $p = 7$.

Single-Point Crossover:

The best search outcome for this specific type of crossover was achieved through bit inversion, as illustrated in the Figure 12. In the 5th generation, a solution to this problem was successfully discovered. The second favorable result was accomplished using mutation with swapping, which was resolved by the 7th generation. The final mutation approach required 11 generations to yield results.

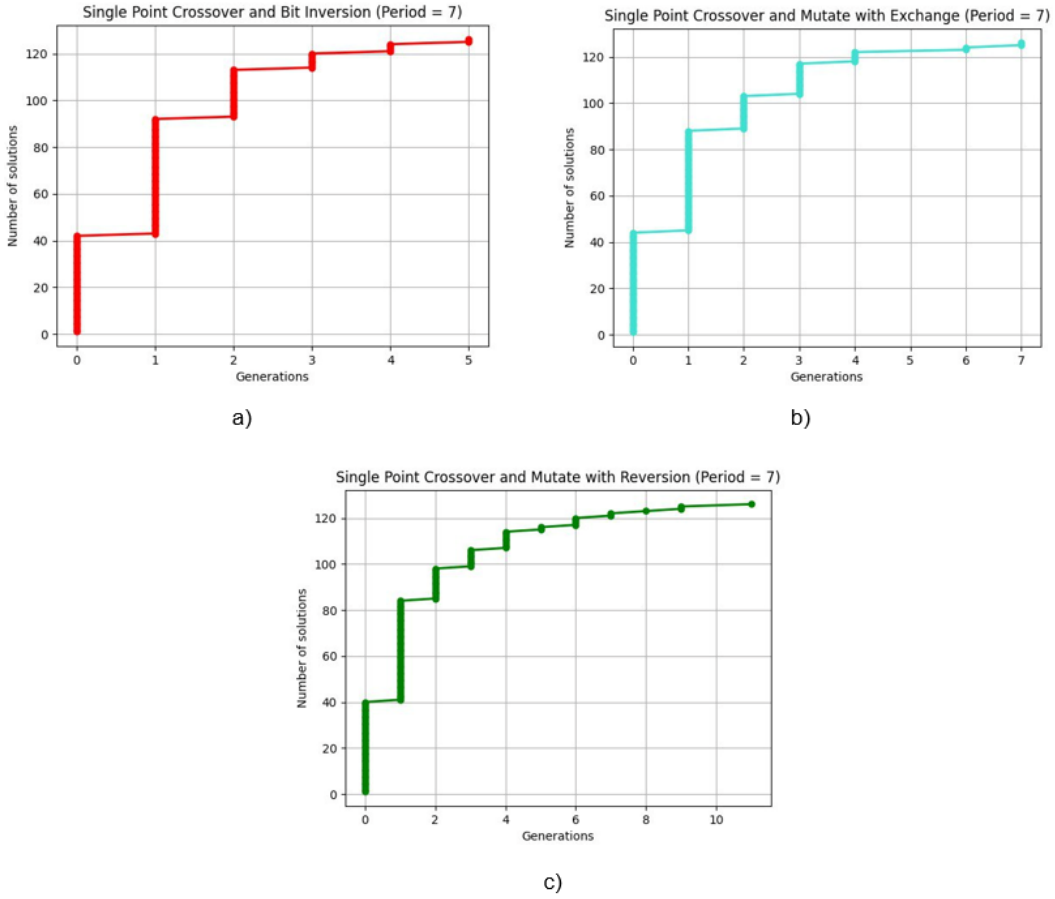

Figure 12. Results of the algorithm of single-point crossover with mutations: (**a**) bit inversion, (**b**) exchange, (**c**) reversion.

K-Point Crossover:

In the case of K = 3 point crossover, the mutation with exchange in the 9th generation, the mutation of bit inversion in the 10th generation, and the mutation with reversal in the 13th generation gave the best result (Figure 13).

The results of uniform crossover are as follows (Figure 14): the mutation was handled in the 9th generation, the mutation with bit inversion in the 253rd generation, and the mutation with exchange in the 399th generation. In this case, it can be seen that for this type of crossover, a mutation with an appeal copes very well for this task.

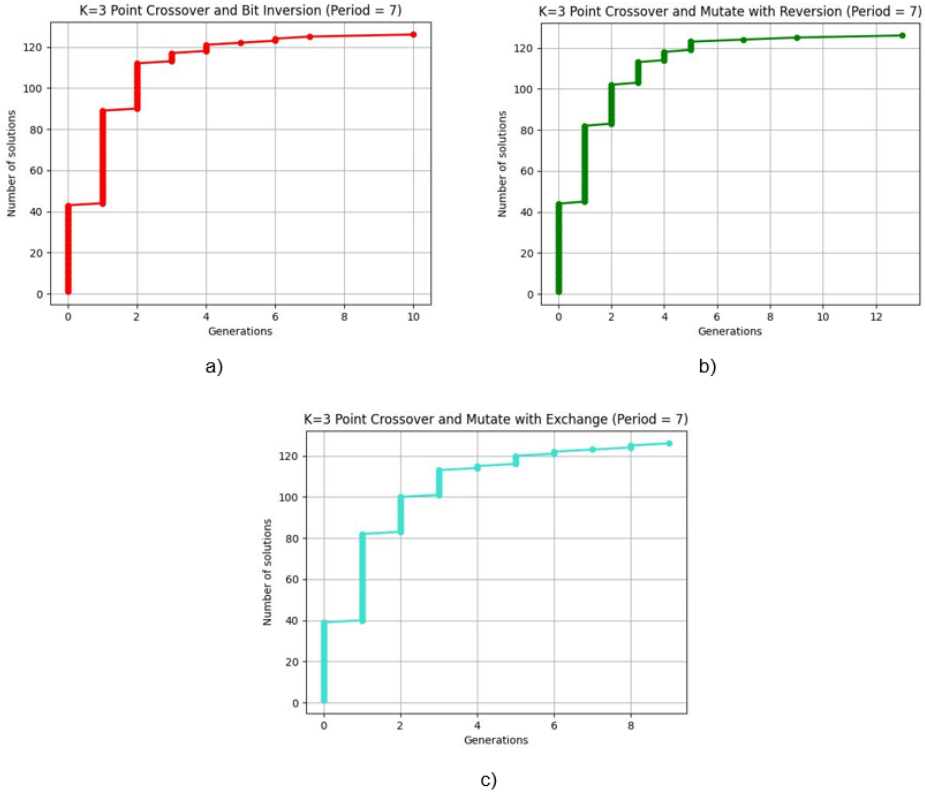

Figure 13. Results of the algorithm of K = 3 point crossover with mutations: (**a**) bit inversion, (**b**) reversion, (**c**) exchange.

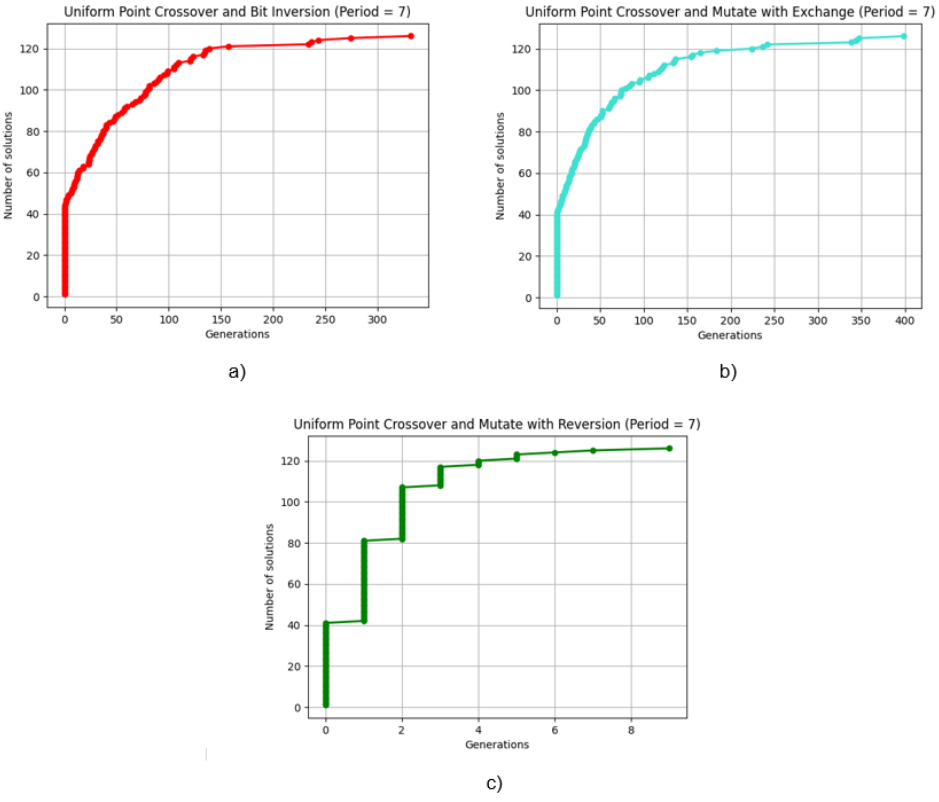

Figure 14. Results of the algorithm of uniform crossover with mutations: (**a**) bit inversion, (**b**) exchange, (**c**) reversion.

5.3. Error Calculation

After we determined the most optimal of all the considered algorithms for finding variations, it was necessary to calculate the errors. We will investigate the previously obtained sequence s_n (n = 100,000) using the logistic map for periodicity. To do this, we will use Formulas (1) and (2) and calculate the error. As mentioned earlier, the minimum calculated error is selected for each period value. Table 2 calculates all the minimum errors for each value of the period from 2 to 11:

Table 2. The results of transforming the previously obtained sequence into a periodic form using the logistic map.

Period	**Solutions (n = 15)**
2	1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
3	1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0
4	1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0

As can be seen from Table 3, in general, the error values decrease with increasing period values. This is due to the property of unpredictability [3]. It is important to note that the error values for periods 2 and 4 are identical.

Table 3. The results of calculating all minimum errors for each period value ranging from 2 to 11.

Period	**MinError**
2	0.04357534482380826
3	0.09115647632270596
4	0.04357534482380826
5	0.018129348944227353
6	0.011829551497279324
7	0.0031728354084733235
8	0.0006835432479136291
9	0.001426824949332663
10	0.0005645921357144535
11	0.00038676431381845375

6. Conclusions

During the research, an efficient method utilizing a genetic algorithm was developed for transforming binary sequences with a size of over 100,000 elements into their periodic counterparts. This method allows for extracting all possible variations of transformation

from the original sequence to its periodic form for a specific chosen period value. An analysis was conducted on all obtained variations using sequences generated with the logistic map. As a result of error calculations for a sequence of size n = 100,000 using a logistic map for periodicity, minimum errors were obtained for each value of the period from 2 to 11. For example, for period 7, the minimum error was 0.0031728354084733235. To optimize the genetic algorithm process, an algorithm called MakePeriod was developed, which yielded results in just 2 generations, as opposed to 2314 previously. The use of the MakePeriod method reduced the number of generations from 2314 to 2, which represents an improvement of 99.91%. The concept of error for nonperiodic sequences was introduced, and these errors were analyzed within the period values ranging from 2 to 11 for sequences with a size of 100,000. As the period value increases, the error value decreases, which in turn is due to the unpredictability property. The study of errors in the interval of period values can also be key to understanding which parameters and conditions can affect the accuracy of the approximation. Furthermore, a comparative analysis of three different crossover methods and three mutation methods was performed. The best results were achieved using single-point crossover with bit inversion, where a solution for a period of 7 was found in just 5 generations. As a result of analyzing the operation of the algorithm of single-point crossover with mutations, it can be noted that all three mutation variants (bit inversion, exchange, reversion) lead to a maximum value of 128. Single-point crossover with bit inversion reached the optimal solution with a period of 7 over 5 generations. This indicates the effectiveness of the algorithm when using these mutation methods in combination with single-point crossover. Crossover with uniform distribution showed the lowest performance, requiring 399 generations to reach a solution. The developed algorithm offers significant potential for application in cases involving large sequences and period values. This research approach presents significant potential for solving complex problems related to the approximation of nonperiodic sequences. The study in this work demonstrates that with an increase in the period value, the accuracy of approximation improves, making this method particularly useful for sequences with long periods. It is also important to note that this work emphasizes the importance of the correct selection of crossover and mutation methods when applying genetic algorithms to this problem, which, in turn, affects the convergence speed and efficiency of the algorithms.

Author Contributions: Investigation, M.Z., M.A., Y.A. and Z.B.; Project administration, M.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research has been funded by the science committee of the Ministry of Science and Higher Education of the Republic of Kazakhstan (grant nos. AP19579370 and BR18574144).

Data Availability Statement: Data is contained within the article.

Acknowledgments: The authors would like to thank the science committee of the Ministry of Science and Higher Education of the Republic of Kazakhstan (grant nos. AP19579370 and BR18574144).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Gleick, J. *Chaos: Making a New Science*; Penguin: London, UK, 2008.
2. Kanter, I.; Butkovski, M.; Peleg, Y.; Zigzag, M.; Aviad, Y.; Reidler, I.; Rosenbluh, M.; Kinzel, W. Synchronization of random bit generators based on coupled chaotic lasers and application to cryptography. *Opt. Express* **2010**, *18*, 18292–18302. [CrossRef]
3. Akhmet, M.; Tola, A. Unpredictable strings. *arXiv* **2020**, arXiv:2006.08523.
4. Castañeda, L.B.; Arunachalam, V.; Dharmaraja, S. *Introduction to Probability and Stochastic Processes with Applications*; John Wiley & Sons: Hoboken, NJ, USA, 2012.
5. Martínez-Ñonthe, J.A.; Castañeda-Solís, A.; Díaz-Méndez, A.; Cruz-Irisson, M.; Vázquez-Medina, R. Chaotic block cryptosystem using high precision approaches to tent map. *Microelectron. Eng.* **2012**, *90*, 168–172. [CrossRef]
6. López-Mancilla, D.; López-Cahuich, G.; Posadas-Castillo, C.; Castañeda, C.E.; García-López, J.H.; Vázquez-Gutiérrez, J.L.; Tlelo-Cuautle, E. Synchronization of complex networks of identical and nonidentical chaotic systems via model-matching control. *PLoS ONE* **2019**, *14*, E0216349. [CrossRef] [PubMed]

7. Jaimes-Reátegui, R.; Sevilla-Escoboza, R.; Pisarchik, A.N.; García-López, J.H.; Huerta-Cuellar, G.; Ruiz-Oliveras, F.; Lopez Mancilla, D.; Castañeda-Hernandez, C.E. Secure optoelectronic communication using laser diode driving by chaotic Rössler oscillators. *J. Phys. Conf. Ser.* **2011**, *274*, 012024. [CrossRef]
8. Akhmet, M.; Fen, M.O.; Alejaily, E.M. A randomly determined unpredictable function. *arXiv* **2019**, arXiv:1910.12758.
9. Robert, L.D. *Introduction to Chaotic Dynamical Systems*; CHAPMAN & HALL CRC: Boca Raton, FL, USA, 2021.
10. Pugacheva, V.; Korotkov, A.; Korotkov, E. Search of latent periodicity in amino acid sequences by means of genetic algorithm and dynamic programming. *Stat. Appl. Genet. Mol. Biol.* **2016**, *15*, 381–400. [CrossRef] [PubMed]
11. Akhmet, M. *Principles of Discontinuous Dynamical Systems*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2010.
12. Kanso, A.; Smaoui, N. Logistic chaotic maps for binary numbers generations. *Chaos Solitons Fractals* **2009**, *40*, 2557–2568. [CrossRef]
13. Akhmet, M.; Fen, M.O. Non-autonomous equations with unpredictable solutions. *Commun. Nonlinear Sci. Numer. Simul.* **2018**, *59*, 657–670. [CrossRef]
14. Mokhtarzadeh, M.; Tavakkoli-Moghaddam, R.; Triki, C.; Rahimi, Y. A hybrid of clustering and meta-heuristic algorithms to solve a p-mobile hub location–allocation problem with the depreciation cost of hub facilities. *Eng. Appl. Artif. Intell.* **2021**, *98*, 104121. [CrossRef]
15. Lambora, A.; Gupta, K.; Chopra, K. Genetic algorithm—A literature review. In Proceedings of the 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), Faridabad, India, 14–16 February 2019; IEEE: Piscataway, NJ, USA, 2019.
16. Wirsansky, E. *Hands-On Genetic Algorithms with Python: Applying Genetic Algorithms to Solve Real-World Deep Learning and Artificial Intelligence Problems*; Packt Publishing Ltd.: Birmingham, UK, 2020.
17. Dagdia, Z.C.; Mirchev, M. Chapter 15—When Evolutionary Computing Meets Astro- and Geoinformatics. In *Knowledge Discovery in Big Data from Astronomy and Earth Observation*; Škoda, P., Adam, F., Eds.; Elsevier: Amsterdam, The Netherlands, 2020; pp. 283–306. [CrossRef]
18. Changat, M.; Narasimha-Shenoi, P.G.; Nezhad, F.H.; Kovše, M.; Mohandas, S.; Ramachandran, A.; Stadler, P.F. Topological Representation of the Transit Sets of k-Point Crossover Operators. *arXiv*, **2017**, arXiv:1712.09022.
19. Gitcho, P.; Wagner, G.P. Recombination Induced Hypergraphs: A New Approach to Mutation-Recombination Isomorphism. *Complexity* **1996**, *2*, 47–43.
20. Hu, X.-B.; Di Paolo, E. An efficient genetic algorithm with uniform crossover for air traffic control. *Comput. Oper. Res.* **2009**, *36*, 245–259. [CrossRef]
21. Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*; Addison Welssey Publishing Company: Reading, MA, USA, 1989.
22. Holland, J.H. Genetic Algorithms. *Sci. Am.* **1992**, *267*, 66–73. [CrossRef]