# GENETIC ALGORITHM FOR FINDING THE KEY'S LENGTH AND CRYPTANALYSIS OF THE PERMUTATION CIPHER

Aleksey Gorodilov, Vladimir Morozenko

*Abstract: In this article we discuss a possibility to use genetic algorithms in cryptanalysis. We developed and described the genetic algorithm for finding the secret key of a block permutation cipher. In this case key is a permutation of some first natural numbers. Our algorithm finds the exact key's length and the key with controlled accuracy. Evaluation of conducted experiment's results shows that the almost automatic cryptanalysis is possible.*

*Keywords: cryptography, cryptanalysis, block permutation cipher, genetic algorithm, data encryption*

*ACM Classification Keywords: I.2 Artificial Intelligence: I.2.8 Problem Solving, Control Methods, and Search - Heuristic methods, E.3 DATA ENCRYPTION Code breaking.*

## Introduction

The main problems in cryptography are the development of reliable cryptographic schemes (a cryptography problem) and the search for new effective methods of deciphering existing schemes (a cryptanalysis problem). A cryptographic approach to secure information implies its transformation which enables it to be read only by the owner of the secret key. The reliability of a cryptographic method of securing data depends on cryptanalysis stability of the used scheme. When we talk about cryptanalysis we assume that we know the cryptographic scheme, but we don't know the key and/or its length. In other words, a cipher breaking problem (a cryptanalysis problem) is a problem of finding only one true secret key among all possible secret keys, i.e. it is a search problem. The search space is large and the criterion of found solution's "quality" is not usually purely formalized.

In this article we focus on the cryptanalysis problem of the block permutation cipher. The secret key of this cipher is a permutation of some first natural numbers with unknown length. To solve the cryptanalysis problem we use genetic algorithms. It seems reasonable to apply genetic algorithms, because they are successfully used in search problems and optimization problems. You can read about solving cryptanalysis problems by genetic algorithms in [Delman, 2005, Lebedev, 2005, Jakobsen, 1995]. But authors of these articles assume that the key's length is known and they use standard methods which do not consider peculiarities of the task or they don't investigate how algorithm's parameters can affect its convergence speed.

Before developing any genetic algorithm, you must choose the proper way to present possible solution as a symbol string and also choose proper main operators – selection, crossover and mutation. The quality of the genetic algorithm should be adjustable. We should have a possibility to change algorithm's parameters such as the size of the population, the number of generations and probabilistic characteristics of main operators.

## A block permutation cipher

In this article we consider the particular cryptographic scheme – the symmetric block permutation cipher. The main idea of this cipher is that we divide the input text into blocks, i.e. into symbol strings with the fixed length equal to $N$, and then symbols in every block are rearranged in accordance with the given permutation

$$P = \begin{pmatrix} 1 & 2 & 3 & \dots & i & \dots & N \\ p_1 & p_2 & p_3 & \dots & p_i & \dots & p_N \end{pmatrix},$$

where $p_i \in \{1,2,3,\dots,N\}$. In other words, symbol at the position $i$ is moved to the position $p_i$. The secret key of that cipher is the permutation $P$. Deciphering is performed using the inverse permutation $P^{-1}$.

## The genetic algorithm

A permutation will play the role of individual in the genetic algorithm, because the solution of the cryptanalysis problem here is also a permutation. At first we suppose that we know the key's length and we must find only the key, i.e. the permutation with the fixed length $N$.

The main problem we must solve is the role of separate genes of an individual. The first simple solution which seems the most evident is to associate separate genes with elements of the permutation $P$, i.e. associate the $i^{th}$ gene with the number $p_i$. Obviously in that case genes will depend on each other. If some gene is equal to $j$, then no other gene of this individual can be equal to $j$, because all numbers between 1 and $N$ are presented only once in the permutation $P$. The dependence of genes results in important limitations to operators of mutation and crossover. We can't use standard operators in this case, because all of them work with a string of independent bits. Such interconnections between genes don't reside in wildlife. This leads to prejudices in the effectiveness of using genetic algorithms. But this association is subconsciously comprehensible and doesn't need any additional computations to form genes.

The alternative solution is to use an intermediate presentation of an individual in a form of an object which can be easily transformed to a permutation. At the same time such object should be presented as a string of independent bits to make applicable mutation and crossover operators. In this case the problem of choosing a proper intermediate presentation can be very complex. In this article we choose the first solution which is the most comprehensible. It means that in the rest part of the article we will regard separate genes as elements of the permutation $P$.

The next problem is how to calculate fitness value for individuals. We suppose that input text is a literary text in Russian. Fitness function is taken from Jakobsen's works [Jakobsen, 1995, Agranovskiy, 2002]. In 1995 Thomas Jakobsen suggested an automatic method for deciphering a simple substitution cipher. In Jakobsen's method the information about frequency sharing of digrams in literary texts is used. Digram is a pair of two neighboring symbols in text. Jakobsen suggested calculating the fitness value as the sum of modules of differences between predefined etalon digram frequencies and real digram frequencies in the ciphertext. Let $T_{ij}$ be the relative frequencies of digrams $(i, j)$ in the text $T$. Then the fitness function would look like

$$W(T) = \sum_{ij} \left| T_{ij} - E_{ij} \right|$$

where $E_{ij}$ is the predefined etalon frequencies of digrams $(i, j)$. Etalon frequencies' matrix $E$ is calculated beforehand under big literary text in Russian. Obviously, the more literary is the text, the less is the fitness value and the "closer" is the found key to the true secret key. It means that fitness value is in inverse proportion with individual's fitness. It is important that fitness function can be not equal to zero even if the ciphertext is deciphered with the true secret key.

So, if we have the ciphertext $S$, then we should perform the following steps to calculate fitness value of the individual $P$.

1. Decipher ciphertext $S$ with the selected key $P$. As a result we would get the text $T = Decrypt_P(S)$..
2. Calculate digrams frequencies $T_{ij}$ in the text $T$.
3. Calculate the fitness value $W(T)$.

Let's focus on the properties of the fitness function $W(T)$. Let $P_1$ and $P_2$ be two individuals and let them differ in only two first genes (obviously they can't differ in the only one gene). Suppose for instance

$$P_1 = \begin{pmatrix} 1 & 2 & 3 & \dots & N \\ p_1 & p_2 & p_3 & \cdots & p_N \end{pmatrix},$$

$$P_2 = \begin{pmatrix} 1 & 2 & 3 & \dots & N \\ p_2 & p_1 & p_3 & \cdots & p_N \end{pmatrix}.$$

Suppose, we have ciphertext $S$. Texts $Decrypt_{P_1}(S)$ and $Decrypt_{P_2}(S)$ deciphered with keys $P_1$ and $P_2$ differ only in symbols which are at positions $p_1$, $p_2$, $p_1 + N$, $p_2 + N$, $p_1 + 2N$, $p_2 + 2N$ and so on. These numbers

form two arithmetic progressions with the difference of $N$. Because texts $Decrypt_{P_1}(S)$ and $Decrypt_{P_2}(S)$ can differ only in these positions, each block with a length of $N$ has no more than 3 digrams which differ in their frequencies. So, when the key's length $N$ is big enough (this is true for real schemes), a small change of an individual would cause a small change of a fitness value.

Also notice that in long literary texts digram frequencies $T_{ij}$ are close to corresponding etalon frequencies $E_{ij}$. That's why in the result of deciphering the text $T$ with the true secret key fitness value would be close to zero. So, if $K$ is the true secret key then the value $W(Decrypt_K(S))$ would be the minimal possible value.

As it was mentioned above we can use a standard crossover operator only if an individual is presented as a string of independent bits. In our case individuals are not strings of independent bits, so we can't use standard operators. We should develop a special crossover operator. The main requirement to the crossover operator is that the result of its execution should give a correct permutation. In this article we suggest the next crossover operator:

1. Enumerate all genes with numbers 1, 2, 3, …, $N$ and scan them in increasing order.
2. Copy gene with the next number from one of the parents, if it is possible.
3. Assign any possible number to gene if genes from both parents are impossible.

Let $P(i)$ be the number which replaces the number $i$ in permutation $P$. So, we have the next crossover algorithm for two parents $P_1$ and $P_2$ and their child $P^*$.

1. Assign a value of Ø to the set *Used*. *Used* is the set of already used gene values. Set the number of current gene $i$ to 1.
2. Define whether numbers $P_1(i)$ and $P_2(i)$ belong to *Used* or not.
3. If only one of numbers $P_1(i)$ and $P_2(i)$ doesn't belong to *Used* then assign it to the $i^{th}$ gene of $P^*$. If both numbers don't belong to *Used* then choose number $P_1(i)$ with probability $p_c$ or choose $P_2(i)$ with probability $(1 - p_c)$. Assign the selected number to the $i^{th}$ gene of $P^*$. If both numbers belong to *Used*, then assign random number from the set $\{1, 2, 3, \ldots, N\} \backslash Used$ to the $i^{th}$ gene of $P^*$.
4. Include the number which was assigned to the $i^{th}$ gene of $P^*$ in the set *Used*.
5. Move to the next gene, i.e. increase $i$ by 1 and go to step 2.

We can use a standard exchange operator as the mutation operator. The idea of exchange operator is very simple. Some two genes of the individual exchange their positions with some probability. To develop the selection operator we use two main classical ideas: the "wheel principle" and the "elite principle" [Mitchell, 1999]. The population's size remains constant because every time new two children are obtained we delete two individuals with the smallest fitness from the population. Finally, we selected the condition of the end of calculations. Calculations stop when era number reaches a fixed number $M$. Because of this condition we can predict time that is needed to complete the calculations or we can define the parameter $M$ so that the algorithm ends after specified time.

## The influence of genetic algorithm's parameters

So, we developed a genetic algorithm for solving the problem of cryptanalysis of the block permutation cipher. We supposed that we know the key's length $N$. Let's discuss the optimal values for parameters. The main parameters which were mentioned early in this article and which influence the algorithm's speed and the solution's quality are:

- the population's size $k$;
- the probability of copying a gene from one of the parents $p_c$;
- the mutation's probability $p_m$;
- the number of generations $M$.

There are no theoretical recommendations on how to choose these parameters. We can only say that the mutation's probability should be small and $p_c$ must be close to 0.5. The number of generations can be chosen depending on time resources. Generally speaking, if the number of generations is the only condition for the end of calculations it must be big enough. The population's size at the start is also important because the population's size is constant in our genetic algorithm and is always equal to $k$. Notice that the population's size has a considerable influence on the complexity of computations and the time required for them.

Conducted experiments show that small population's size gives bad results because the algorithm quickly finds a local minimum point of the fitness function and all further populations are formed in its vicinity. According to these experiments if key's length equals to 10 then the optimal value for population's size is between 15 and 17. Changing the mutation's probability in a reasonable range doesn't have a significant impact on the results. It can be partially explained that we used a specific crossover operator which already implies some mutation of individuals. Indeed, when the next gene cannot be copied from one of the parents it is simply chosen randomly, i.e. child doesn't "resemble" any of his parents in this gene. The optimal number of generations significantly depends on the key's length $N$. If $N$ is small, a big number of generations isn't effective because after some populations algorithm finds a local minimum and all further calculations don't lead to a better result. Possible solutions' space grows with the growth of the key's length. It requires more generations to find the best solution.

At the end of the genetic algorithm's work we obtain a text called decrypted text. That decrypted text doesn't considerably differ from the starting open text. Usually we can restore separate symbols from the context. So, algorithm doesn't completely solve the cryptanalysis problem but greatly helps to decrypt the given text.

## Finding the key's length

The genetic algorithm described above works on the assumption that the key's length $N$ is known. The result of its work is the permutation which corresponds to the best individual from the final population. Because this permutation has the minimum value of the fitness function $W(T)$ genetic algorithm also calculates the value $F(N)$ by the given key's length $N$. $F(N)$ indicates the minimal found value of the fitness function.

Let's now focus on the key's length finding problem from the given ciphertext. Let $N$ be the supposed key's length and $L$ be the true key's length. If $N$ is equal to $L$ then we can expect that the permutation found by the genetic algorithm is close to the searched permutation and the fitness function corresponding to this permutation would have a relatively small value.

Let the true secret key be the permutation

$$K = \begin{pmatrix} 1 & 2 & \cdots & L \\ p_1 & p_2 & \cdots & p_L \end{pmatrix}$$

Consider the key with length $2 \cdot L$ of a form

$$K' = \begin{pmatrix} 1 & 2 & \cdots & L & L+1 & \cdots & 2 \cdot L \\ p_1 & p_2 & \cdots & p_L & L+p_1 & \cdots & L+p_L \end{pmatrix}$$

In $K'$ first $L$ columns are absolutely equal to first $L$ columns in permutation $K$. The rest columns are obtained from the first $L$ columns by adding $L$ to all numbers. We can correctly decipher the ciphertext with the key $K'$ because both keys $K$ and $K'$ identically transform any text. It means that when genetic algorithm searches the key with the length $2 \cdot L$ it would find a permutation which is close to $K'$. At the same time a small fitness value would correspond to the found permutation. Therefore the value $F(2 \cdot L)$ would be relatively small. Similar statements are true for all keys have length divisible by $L$. In other words, if we pass the ciphertext and the key's length $N = m \cdot L$ (where $m$ is a natural number) to the input of genetic algorithm then we can expect a permutation with a small fitness value at the output. Otherwise, if $N$ isn't divisible by $L$ then there is no permutation with the length $N$ which are close to the key $K$. In this case fitness value and value $F(N)$ are expected to be relatively large. So, value $F(m \cdot L)$ is expected to be much smaller than $F(N)$ for all N which are not divisible by $L$.

So, we offer the following algorithm for the finding of the key's length. At first, choose some initial value $N_0$ for the key's length. The value $F(N_0)$ is calculated using the genetic algorithm. At the next step we increase the supposed key's length by 1 and calculate value $F(N_1)$ (where $N_1 = N_0 + 1$) using the genetic algorithm and so on. At the $i^{th}$

step we calculate value $F(N_i)$ where $N_i = N_0 + i$. This algorithm continues while $N_i < N^*$ where $N^*$ is a predefined value (is an upper bound). As a result of this process we have a sequence of numbers

$$F(N_0),\ F(N_0 + 1),\ F(N_0 + 2),\ \ldots,\ F(N_0 + i),\ \ldots,\ F(N^*).$$

In this sequence we can mark out small values having positions expected to form an arithmetical progression with the step $L$. The value of $L$ is the searched true key's length.

Notice that we can use a small population's size and a small number of generations while searching the key's length to save time resources. When the length of the key is found we can restart genetic algorithm with big values for parameters to find the secret key.

## An example of algorithm's work

As the input text we took a fragment of literary text in Russian:

> *Было тихое летнее утро. Солнце уже довольно высоко стояло на чистом небе, но поля еще блестели росой, из недавно проснувшихся долин веяло душистой свежестью, и в лесу еще сыром и не шумном, весело распевали ранние птички. На вершине пологого холма, сверху донизу покрытого…*

In experiment this text was encrypted using the block permutation cipher with the key's length equal to 10. In the result we had the next ciphertext:

> *т иолоеыхБенеуеттл оСлц. еонрд оожелув ывско оноьляонтоас отм ис ч,н бепеонщеебя лл оил отессрези е, дйнол рсноновоасхядшиов уевяон иллтсо шисуйдтсь,же еювелс в е уирыю сиемщушмое мнн есл вер о,авл пе сиаеи тнниапрН ави.ек ч еплиношорх омгоаолгрех свд у, упкизрноо о ог т ы…*

From this meaningless set of symbols it is difficult to restore the start text manually. Then this ciphertext was passed to the input of the genetic algorithm in order to find the secret key's length. The range of the supposed key's lengths was [4; 34], i.e. $N_0$=4, $N^*$=34. For all supposed key's lengths we calculated corresponding values $F(4)$, $F(5)$, …, $F(34)$. Figure 1 shows the reverse values $11/F(4)$, $1/F(5)$, …, $1/F(34)$. The higher point at the diagram the closer its abscissa to the true key's length $L$ or to a value, divisible by $L$.

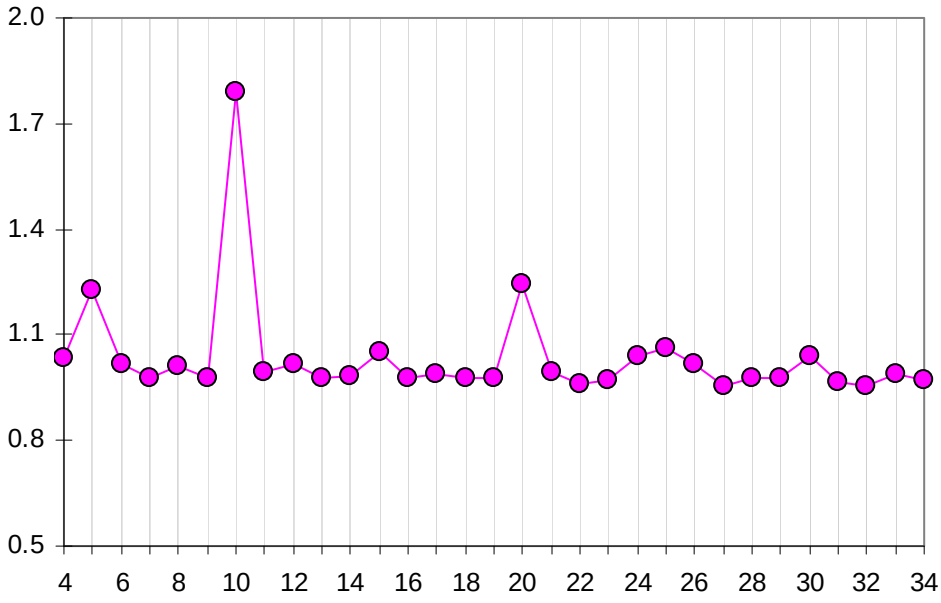


*Fig1. Diagram of the dependence 1/F(N) where N is the supposed key's length.*

On the figure we can see the maximal "spike" of the fitness function which corresponds to the length 10. Notice that 10 is the true secret key's length. The next noticeable local maximum is at $N$=20 which corresponds to the double value $2 \cdot L$. Further there are no noticeable "spikes" but at $N$=30 (which equals to the triple value $3 \cdot L$) we can see a local maximum.

Results of this experiment conform well to the theory. So, in this experiment we can affirm that the secret key's length $L$ is equal to 10. Then genetic algorithm was restarted to find the secret key. We used the following values for algorithm's parameters: the population's size – 15, the number of generations – 30 and the mutation's probability – 0.2. In the result we obtained the next text:

> *Былохти ое лет еенутро. нолСце ужеvдо ольно оысвко сто лояна чис омт небе он, поля щееблестери лосой, нз иедавноопр снувши сяхдолин леяво душийтос свежеютьс, и в уесл еще смроы и не нумшом, веоелс распеиалв раннпе итички. На вершино пелогоголхо ма, свурхе донизо пукрытог о…*

As we can see, algorithm didn't completely solve the problem because the result text isn't identical to the input text. But we can see that this text is close to the input text. For example, we can see several words which are equal to the corresponding words in the input text: "было", "уже", "поля", "небе" and so on. It seemed easy to finish the decryption manually using this result text.

Moreover, we can compare the true secret key and the result of the genetic algorithm. The true secret key is the permutation

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 10 & 8 & 5 & 6 & 2 & 1 & 3 & 9 & 4 & 7 \end{pmatrix}.$$

The result of the genetic algorithm is

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 10 & 8 & 5 & 6 & 9 & 1 & 3 & 2 & 4 & 7 \end{pmatrix}.$$

These permutations differ in only one pair of values. It seems easy to look through a small number of variants and to find the true secret key manually.

Certainly the genetic algorithm for finding the secret key described in this article doesn't completely automate the text decryption but it makes the decryption faster. After using this algorithm we can easily finish the decryption process manually.

## Bibliography

[Delman, 2005] Delman B. GeneticAlgorithms in Cryptography // A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering. New York, 2004.

[Лебедев, 2005] Лебедев А. В криптографии мы способны конкурировать / ЛАН Крипто, 2005.

[Jakobsen, 1995] Jakobsen T. A Fast Method for the Cryptanalysis of Substitution Ciphers, 1995.

[Agranovskiy, 2002] Аграновский А. В., Хади Р.А. Практическая криптография: алгоритмы и их программирование. М.:СОЛОН-Пресс, 2002.

[Mitchell, 1999] Mitchell M. An Introduction to Genetic Algorithms. Fifth printing. Cambridge, MA: The MIT Press, 1999.

## Authors' Information

*Aleksey Gorodilov – University Undergraduate, Computer Science Department, Perm State University, Bukireva street 15, Perm 614990, Russia; e-mail: gora830@yandex.ru*

*Vladimir Morozenko – Assistant Professor, Computer Science Department, Perm State University, Bukireva street 15, Perm 614990, Russia; e-mail: v.morozenko@mail.ru*