The timer inremental compression of data and information

RUSLAN SKURATOVSKII
Department of computer science
Igor Sikorsky Kiev Polytechnic Institute
Kiev, UKRAINE

VOLODYMYR OSADCHYY
Department of computer science
IT-GRAVITY-VO, Inc. Orlando
Florida USA,

YEVGEN OSADCHYY
National University named by Taras Shevchenko
Kiev, UKRAINE

Abstract: — The ability to find short representations, i.e. to compress data, is crucial for many intelligent systems. This paper is devoted to data compression and a transform-based quantitative data compression technique involving quick enumeration in a unary-binary time-based numeral system (NS). The symbols comprising the alphabets of human-computer interaction languages (HCIL), which are used in an informational message (IM), are collected in primary code tables, such as the ASCII table. The statistical-oriented data compression method using unconventional timer encryption and encoding information are proposed by us. It was constructed probability - discrete model of the set of character sequences and characterized some probabilistic algorithms associated with the recovery of text by its public key and its cipher. We find the possibility of parallel implementation of this method by building a block of timer tags. The necessary estimations of complexity are obtained. The method can be used to compress SMS messages. Probabilistic-statistical analysis and evaluation of their effectiveness are obtained.

Key-Words: - Block coding, archiving, data compression algorithm, timer coding.

Received: February 18, 2020, Revised: July 10, 2020. Accepted: July 31, 2020. Published: August 16, 2020.

1 Introduction

The main tasks of character-coding are the creation of methods and tools for storage, compression, protection, etc. To date, many theoretical binary coding limits have already been practically achieved, but there are unused opportunities. First of all, they contain the so-called "timer" encoding. It has already been used in a binary Central Committee due to a partial implementation - the introduction of a timer. The aim of the work is to prove that the process of restoring text using a statistical text generator *Gs*, which operates according to a deterministic algorithm, can be unambiguously mapped onto the process of recording timer marks with minimal complexity. Thus, we one-to-one associate a text, or some numerical sequence, for example, a generalized Fibonacci sequence [1], over the alphabet *X* and its code in the timer coding system, with the alphabet *Y*, which, in particular, is the alphabet for unit calculus system, which is actually set by timer marks [1,2]. The set of all tuples of lengths *n* will be denoted by X^n [7]. The effectiveness of statistical analysis with subsequent data compression is proved.

2. Review of the results of predecessors

As shown by studies of data compression programs based on classical compression methods, their efficiency is already close to the compression limits defined by the classics of coding theory. Using concrete examples, [1, 2, 3, 4, 15] showed that compression after Shannon–Fano methods and their successors [3, 4] repeated lossless encoding compresses by several times the magnitude of a number properly corresponding to the CC of the incoming IM, while lossy encoding yields compression by tens of times. The researchers did not address the efficiency of BD compression by these methods, especially when the IM's primary coding has entropy. But based on our preliminary analysis of classical compression methods' capabilities, the achieved compression/decompression ratio of such IMs will not improve.

A probabilistic-statistical analysis and estimation of the efficiency of a statistically oriented generator is compared with the over-combinatorial generator *Gt* described in [1–3]. Coding with labels on the tape of a machine record was started by A. Turing [3]. To compress and protect information, F. Bardachenko in [4] in the form of timer masquerading. We focused on the results and evidence of the effectiveness of the statistically trained text generator we proposed [2, 4, 8, 10, 13]. In this work we continue the investigation of author [10, 14, 15].

3. Timer coding in language communication

In accordance with the provisions of [7, 8, 13], the quantitative value of a binary number in one bit, displayed through the direct access memory in ASCII, determines the sequence of quantitative values from 1 ... to $256_{(10)}$ records that can be used using 8 bits (cells). A similar number of cells will be needed when using a binary-single midrange $NS_{(2-1)}$. This is shown in the results table. It will be a combination of characters, but from 8 records for 32, the number of one bit memory cells is: 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000001010000000000000000000000000. The algorithm for generating such a sequence is not binary complexity, but shift “1”. The sequence of actions should be increased up to 8 times.

4. Main result

Recall the idea of an autoencoder is to use a descriptive map f_0, which in our case is timer stamp, to project input data x on a shorter residual description r, from which a feature map f can reconstruct it. For instance, consider a string of the form $x = 1^n 0 y$ where $1^n = 11\ldots1$ (n). Then, a descriptive map could be a function computing the number n of initial ones and copying y to the residual description, $f'(x) = \langle n, y \rangle =: r$. Such a description map can be used for compression of our timer stamp. In this case an attribute of string is numbe of 1 in srting. But in first stage of timer stamp generation it is time of work of text generator. Our main purpose generate a timer stamp with maximal data compression and minimal time. But due to the residual description a timer stamp can be additionally compressed.

To control a generation of texts it is used regular binary counter. This is a regular binary counter, the frequency of which is synchronized with the reference time, and the readings are displayed in the alphanumeric notation of the language of time intervals. It is used for: timer interruptions, computational paralleling, processing distributions, and so on. Today, for the creation of blockchain, crypto currency, electronic money, internet of things and many other things we are trying to solve the problems of distributed and remote processing of information.

Definition. The text generator statistically oriented (Gs) is a generator that implements the necessary series of frequency distribution $\wp$, where $f_i > f_j$, characters with ordering of numbers $n_i < n_j$, where n_i — number of the symbol in the received order $\wp$.

Probabilistic-statistical analysis and estimation of the efficiency of a statistically oriented generator Gs are compared with the over-combinatorial generator described in [1,2]. The calculated speed of archiving data is the largest among the world's analogs. The time complexity $O(n^2)$ from which, taking into account the frequency of the generator, it is V easy to obtain a snooze time.

Definition. The text generator with alphabetical ordering (Gp) is a generator that sequentially implements texts, generating characters in alphabetical order, that is, not taking into account the frequency of appearance of these symbols in a certain kind of texts.

Theorem 1. There is a bijective mapping, and for the construction of the mapping, no sequential generation of all preceding texts is necessary for the given text, between the set of binary numbers and the set of timestamps of these numbers.

Indeed, the timestamp of a given binary number *a,* and hence of the corresponding text, can be obtained from formula

$$t = \frac{N_2(a)}{V}. \tag{2}$$

Where $N_2(a)$ is a binary number that corresponds to the code of the given text (the weight of the text for a given symbol generation order); V — frequency of the generator,

$$|N_2(a)| = Wh_p(T),$$

where $Wh_p(T)$ is the complexity of the generation (weight) of the text T for a certain linear ordering. For a statistically oriented generator with a symbolic order $\wp$ the value of the timestamp is given by

$$t(\mu_0) = \frac{Wh_s(T)}{V} \quad . \tag{3}$$

Knowing the weight of the text, you can divide it into k parts. For each part, calculate the weight and calculate the value of the timestamp $t(\mu_i)\ 1 \le i \le k$. Thus, a timestamp system $y = t(\mu_\Sigma)$ by calculating the sum of the series, rather than generating a list of multiple hyperwords in accordance with the lexicographic ordering obtained $\wp$.

Elementary random events will be considered symbols of the text that reach values from the alphabet A, and since the product, sum, the composition of random variables remains a random variable, then we consider a random variable

$$T = \prod_{i_j=1}^{n} \xi_{i_j} = \prod_{j=1}^{n} a_j .$$

A random variable is a measurement function χ, which maps from in . That is, it is necessarily a really significant amount. Therefore, we introduce a one-to-one correspondence of the entire alphabet with a subset of natural numbers capacity $|X|$.

Also we can consider the generated text as a random element - a measurable function that acts with Ω in "Any abstract space." Then "random event ξ " is: $\{\ \xi \mid \chi(\xi) < \text{Const}\ \}$.

Definition. The number of the symbol $s \in A$ for a given ordering $\wp$ there is his decent number $n_s \in$ in this ordering, which is carried out for a statistical generator Gs according to the probabilities characteristic of the symbols of the selected text. Note that the weight of the symbol will be its number n_i in the frequency ordering, such, that if $f_i > f_j$, then $n_j > n_i$.

The text symbols have relative frequencies $f_s,\ s \in A$, where A is alphabet of symbols with T by which we order them by the drop in frequency. Recall that the generator generates symbols in a given descending order sequentially by bit. The time and result of the work is uniquely determined by the timestamp.

Definition. The text generator statistically oriented (Gs) is a generator that implements the necessary series of frequency allocations $\wp$, where $f_i > f_j$, characters with ordering of the numbers $n_i < n_j$, where n_i is the symbol number in the received order $\wp$.

Note that in the case of a parallel block generation of a whole text of length n simultaneously, we have the complexity

$Wh(T) = \sum_{i=1}^{n} n_i |X|^{n-i}$ and the corresponding value of the **timestamp** calculated using the formula:

$$\varphi(a_5N^5 + a_4N^4 + a_3N^3 + a_2N^2 + a_1N + a_0) =$$
$$= \varphi(c_2N^2 + c_1N^1 + c_0) \circ \varphi(d_2N^2 + d_1N\ + d_0),$$

$$a_{n+m}N^{n+m} + \ldots + a_1N + a_0 = (c_nN^n + \ldots + c_0)(d_mN^m + \ldots + d_0),$$
$$a_{n+m+1} = c_n,\ a_{n+m} = c_{n-1},\ \ldots,\ a_m = d_m, \ldots, d_1 = a_0.$$

The text symbols $f_s,\ s \in A$ have relative frequencies, where A is the alphabet of symbols with T by which we order them by the frequency drop. Recall that the generator generates symbols in a given descending order sequentially by bit. The time and result of the work is uniquely determined by the timestamp.

Generating text is as follows: first, it sorts through all the texts of length 1, then length 2, then 3, etc. And when the timer counter works, it stops the work without using the previous texts. Using block coding [3], we generate texts of length n at a time,

without wasting time on generating and sorting out the texts of smaller length.

Definition. The timer generator G_T, in the general sense, is an instrument that implements the count (increment) in the selected system of counting (s.c.) with a constant frequency and sampling. In the quality of system of notation can be a non-positional system of notation, and for a timestamp a unary code can be used, similar to the Rice code, which is based on the non-position unitary calculus system [4]. G_T will be used to stop the generator code text at the right time, when its output will be restored to the initial text.

Let $T=(a_1...a_n)$ is a random text with fixed the distribution of probability $p\left(a_i=s_{i_j}=f_j, j\in(1,...,N)\right), \sum_{j=1}^{N} f_j=1, f_j\geq 0$. Then the conditional mathematical expectation of the complexity of restoring the text is denoted by $E\left(Wh_S(T)\right)$.

Let $ET=\sum_{i=1}^{n} Wh(s_1)P(a_1=s_1)$ is the expected weight of the text from this branch of knowledge.

Theorem 2. The expected complexity of the work of a statistically oriented text generator is less than the expected complexity of the simple test generator found in [2].

We find the formula of the type of conditional mathematical expectation and try to derive the general formula

$$Wh\left(s_{i_1}\right)=\left(n_{i_1}-1\right)(N)^{n-1}+\sum_{i_2=1}^{N} P\left(a_2=s_{i_2}\right)Wh\left(s_{i_2}\mid a_1=s_{i_1}\right),$$

where

$$Wh\left(s_{i_2}\mid a_1=s_{i_1}\right)=\left(n_{i_2}-1\right)(N)^{n-2}+$$
$$+\sum_{i_3=1}^{N} P\left(a_3=s_{i_3}\right)Wh\left(s_{i_3}\mid a_1=s_{i_1}, a_2=s_{i_2}\right)$$

and n_{i_2} (n_{i_2}) is the number of symbol, standing on the first (second) position in the desired text, which is determined in the order that the text generator has. This is the mathematical expectation of the weight for the variant, when the first character is already set correctly and the generator is looking for the correct variant for the 2-*nd* character. Note that on the right side we have the conditional weight (conditional mathematical expectation) of the *i*-th symbol, which depends on the previous symbols. Here recursively defined functions are used.

$$Wh\left(s_{i_3}\mid a_1=s_{i_1}, a_2=s_{i_2}\right)=\left(n_{i_3}-1\right)(N)^{n-3}+$$
$$+\sum_{i_4=1}^{N} P\left(a_4=s_{i_4}\right)Wh\left(s_{i_4}\mid a_1=s_{i_1}, a_2=s_{i_2}, a_3=s_{i_3}\right) \quad (4)$$

And so on we do recursively transformations for symbols s_{i_4} at known a_3 and the same for the subsequent. The last expression is $Wh\left(s_{i_n}\mid a_1=s_{i_1},...,a_{n-1}=s_{i_{n-1}}\right)$ is a mathematical expectation of the weight of the remainder of the text. It is easy to understand that $Wh\left(s_{i_n}\mid a_1=s_{i_1},...,a_{n-1}=s_{i_{n-1}}\right)=n_{i_n}$, furthermore $Wh\left(s_{i_n}\mid a_1=s_{i_1},...,a_{n-1}=s_{i_{n-1}}\right)\leq n_{i_n}$ since in addition to the serial number, the generator still knows the previous symbols from T, therefore having the frequencies of diagrams and trigrams T the generator will decide on the choice of the correct symbol on the nth position on average before

$$Wh\left(s_{i_{n-1}}\mid a_1=s_{i_1},...,a_{n-2}=s_{i_{n-2}}\right)=\left(n_{i_{n-1}}-1\right)N+$$
$$+\sum_{i_n=1}^{N} P\left(a_n=s_{i_n}\right)n_{i_n}=\left(n_{i_{n-1}}-1\right)N+\sum_{i_n=1}^{N} f_{i_n}n_{i_n}.$$

Moving on in the reverse order, we obtain and collapse the formulas using previous formula:

$$Wh\left(s_{i_{n-2}}\left|a_1=s_{i_1},...,a_{n-3}=s_{i_{n-3}}\right.\right)=\left(n_{i_{n-2}}-1\right)N^2+$$
$$+\sum_{i_{n-1}=1}^{N}P\left(a_{n-1}=s_{i_{n-1}}\right)\times$$
$$\times\left(\left(n_{i_{n-1}}-1\right)N+\sum_{i_n=1}^{N}f_{i_n}n_{i_n}\right)=\left(n_{i_{n-2}}-1\right)N^2+$$
$$+\sum_{i_{n-1}=1}^{N}f_{i_{n-1}}\left(\left(n_{i_{n-1}}-1\right)N+\sum_{i_n=1}^{N}f_{i_n}n_{i_n}\right)=$$

$$=\left(n_{i_{n-2}}-1\right)N^2+N\sum_{i_{n-1}=1}^{N}f_{i_{n-1}}n_{i_{n-1}}-$$
$$N\sum_{i_{n-1}=1}^{N}f_{i_{n-1}}+\sum_{i_{n-1}=1}^{N}f_{i_{n-1}}\left(\sum_{i_n=1}^{N}f_{i_n}n_{i_n}\right)=$$
$$=\left(n_{i_{n-2}}-1\right)N^2+N\sum_{i_{n-1}=1}^{N}f_{i_{n-1}}n_{i_{n-1}}-$$
$$-N+\sum_{i_n=1}^{N}f_{i_n}n_{i_n}=$$
$$=\left(n_{i_{n-2}}-1\right)N^2+N\sum_{i_{n-1}=1}^{N}f_{i_{n-1}}n_{i_{n-1}}+$$
$$+\sum_{i_n=1}^{N}f_{i_n}n_{i_n}-N.$$

And, consistently, we substitute:

$$Wh\left(s_{i_{n-3}}\left|a_1=s_{i_1},...,a_{n-4}=s_{i_{n-4}}\right.\right)=\left(n_{i_{n-k}}-1\right)N^3+$$
$$+\sum_{i=1}^{N}(f_{i_n-2}((n_{i_{n-2}}-1)N^2+N\sum_{i_{n-1}=1}^{N}f_{i_{n-1}}n_{i_{n-1}}+\sum_{i_n=1}^{N}f_in_{i_n}-N)$$
$$=(n_{i_{n-3}}-1)N^3+N^2\sum_{i=1}^{N}f_{i_{n-2}}(n_{i_{n-2}}-1)+$$
$$+N\sum_{i=1}^{N}(f_{i_{n-2}}\sum_{i=1}^{N}f_in_{i_{n-1}})+\sum_{i=1}^{N}f_{i_{n-2}}\sum_{i_n=1}^{N}f_in_{i_n}-N\sum_{i_n=1}^{N}f_{i_{n-2}}=$$
$$=(n_{i_{n-3}}-1)N^3+N^2\sum_{i=1}^{N}f_{i_{n-2}}(n_{i_{n-2}}-1)+$$
$$+N\sum_{i=1}^{N}f_in_{i_{n-1}})+\sum_{i_n=1}^{N}f_in_{i_n}-N=$$
$$=(n_{i_{n-3}}-1)N^3+N^2\sum_{i=1}^{N}f_{i_{n-2}}n_{i_{n-2}}+$$
$$+N\sum_{i=1}^{N}f_in_{i_{n-1}}+\sum_{i_n=1}^{N}f_in_{i_n}-N-N^2.$$

Let's analyze the compression ratio. Since the value of expression $\sum_{i=1}^{N}f_in_i$ is minimal when the frequencies $f_1\geq f_2\geq...\geq f_N$ answer ordered in descending weight $n_1\leq n_2\leq...\leq n_N$, then according to Chebishov's inequality [17], this weight is the smallest. It is clear that this value is less than the complexity of the work $Wh(T)=\sum_{i=1}^{n}n_jN^{n-j}$ a conventional serial text generator, which proves the effectiveness of a statistically-oriented generator.

The end of the proof.

We denote by L_{out} this is the sum of the lengths of the codes of words (or blocks of words) in the new alphabet, that is $L_{out}=\sum_{i=1}^{N}l\left(C\left(w_i\right)\right)$. The sum of the lengths of the text characters in the original alphabet, in the case of block coding not symbolic but will be denoted by $L_{in}=\sum_{i=1}^{N}l\left(w_i\right)$, where $l\left(w_i\right)-$ is the length of the *i-th* block of a text. The formula for the compression index gives the following result

$$k=\frac{\sum_{i=1}^{N}l\left(w_i\right)p_i}{\sum_{i=1}^{N}l\left(C\left(w_i\right)\right)p_i}=\frac{N\cdot n\cdot p_i}{N\cdot p_i}=\frac{n}{1}=n.$$

Thus, the compression index of timer coding is equal to *n*.

5. Data compression through methods of improved quantitative compression in a unary-binary numeral system (Ns)

In order to more fully demonstrate the benefits of advanced quantitative compression methods for BD, we will begin a comparison with classical methods, using the quantities of the simplest integers as examples. There is a bijective mapping which is established in **Theorem 1**, and for the construction of the mapping, no sequential generation of all preceding texts is necessary for the given text, between the set of binary numbers and the set of timestamps of these numbers. Before comparing their quantities, we check the state of memory cells (MC). It is known that memory is formatted before read/write operations are performed on a range of memory. Any kind of memory (M) can be reduced to the simplest form: linear memory. The state of M and its constituent MCs can be defined as existent or non-existent.

Amount of data in NS_i at $1 \le i \le n$					
NS_{10}	NS_1	NS_2	$NS_{2\text{-}4}$	$NS_{2\text{-}8}$	NS_{256}
256	1	11111111	11111111 11111111	11111111 11111111 11111111 11111111 11111111	256
...	...	...	...	...	...
8	1	11110	1111 0	11110	
7	1	1111	1111	1111	
6	1	1110	1110	1110	
5	1	111	111	111	5
4	1	110	110	110	4
3	1	11	11	11	3
2	1	10	10	10	2
1	1	01	01	01	1
Time	$256\cdot\nabla_t$	$2^8\cdot\nabla_t$	$2^4\cdot\nabla_t$	$2^3\cdot\nabla_t$	$1\cdot\nabla_t$

Table 2 Diagram of the time spent.

Programmatically, this looks like a corrupted MC, which is avoided via formatting/not included in M. In a binary DC, it is assumed that a single-bit MC may be in one of two states: "0 / 1". Their combinations give rise to all the quantitative variety of the IM, which can fit into the ranges of its M. However, it is known that an MC's "0" state does not actually have a magnitude, but instead is only the magnitude's indication a number's lower-order digits. Therefore, a MC of binary M potentially contains a superfluous quantity. A question arises. Why store something superfluous in M? Indeed, a single-bit MC potentially open to padding with a number of "0/1"s can always be unambiguously updated with them. Diagram of the time spent by the generator (shift processor) to count the amount in memory M of 8 bits for different NS. We denote by ∇_t the time interval of the generator pulse in Table 2. Using the reasoning above, we attempt to confirm the need for the existence of $NS_{1\text{-}2}$, and not solely $NS_{2\text{-}1}$ of arithmetic and any other numeral system designed to identify the magnitude of the positional number. We will show its advantage in compressing the magnitude of the simplest numbers in Table 1.

Size memory in bits M	Absolute valuation of a number		
	$NS_{2\text{-}1}$	$NS_{1\text{-}2}$	NS_{10}
One MC	0 1	- 1	0 1
Two MCs	00 01 00 01 10 00 01 10 11	10 110 1110	00 00 00 00 01 00 00 00 01 02 00 00 01 02 03

Table 1. Absolute valuation of a number in bits (positions) of linear M.

As can be seen from Table 1, as a number's bit-length increases, the compression ratio of the representation of the same quantity of symbols in numbers in $NS_{1\text{-}2}$ changes more quickly than it does for numbers in $NS_{2\text{-}1}$ and NS_{10}.

In subsequent iterations of compression, this behavior continues thanks to a more complex implementation of the counting algorithms. Accordingly, we restrict ourselves to the result of representation of the second iteration of compression for $NS_{1\text{-}2}$. This result is presented in Table 3.

Bit width in M	Representation of the magnitude of a number	No. in NS_{10}
	NS_{1-2}	
One MC	-	0
	1	1
Two MCs	10	01
	*10	02
	**10	03

Table 3. Here, the * symbol denotes a MC that can be filled with any digit/symbol in NS_{2-i}, where $1 \le i \le 2^n$. For clarity, in Figure 1 we present a graphic representation of the relationships identified in Tables 1 and 2.

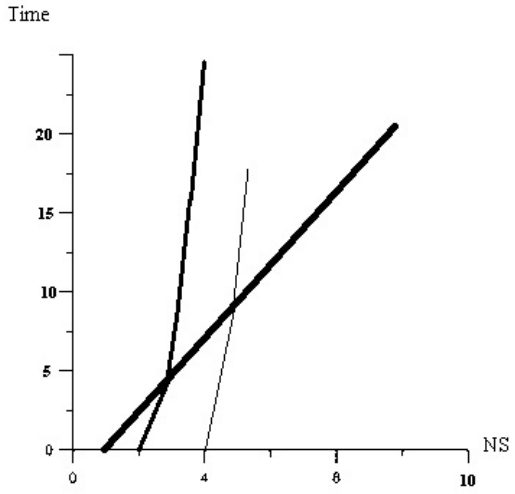

Fig. 1. Efficiency of advanced quantitative compression methods (thick line corresponding to NS_{1-2} and classical methods for NS_{2-i}, where $1 \le i \le 2^n$. A thin line corresponds to compression using a decimal number system

Compression using a single binary system (NS_{1-2}) is more effective because any number is recorded with a single timer label. Compression performance using a timer label based on NS_{1-2} is greater since the number of iterations in a single number system and, therefore, speed is 20% higher than that of a binary number system due to the fact that there is no transfer of to the higher order digit. In a binary counter, the time for counting unit shifts, where is the digit in the number. In the 2-number system, there is a transfer of the discharge to the senior digit when a maximal number of bits, i.e. number of the form 11111 is reached. Since prefix Kolmogorov complexity prefix Kolmogorov complexity satisfy a condition

$$K(f^*) \le l(x) - l(f) + O(\log l(x))$$

f is a attribute of x, $l(x)$ is length of plain text, $l(f)$ is length of attribute of x. Then iterated compression can be applied [18].

6. Conclusion

For quantification, a data compression index was applied. The presented results of the study are sufficient to understand the proposed compression methods. It should only be noted that the possibility of iterative data compression through a continuous algorithm may lead to compressing data to the size of the TS, which can be a single CC. The time required to restore a compressed IM is a secondary objective. It can be accelerated, for example, using parallel high-speed $NS_{(1-2)}$ counters implemented as processors. The main objective of this publication is to study the possibility of compressing information using an alternate (improved) quantitative method based on a transform-based timer-based coding technique.

References

[1] Skuratovskii R., Trembovetska O. Application of discrete structures and numerical sequences in block codes. Naukovie Visti KPI, n.6, 68-75, 2014.

[2] Skuratovskii R.V. The method of fast timer encoding of texts. // Cybernetics and System Analysis, 49 (1): 154-161, 2013.

[3] *Douglas Lind, Brian Marcus*: An introduction to symbolic dynamics and coding, Cambridge University Press 1995.– 490 P.

[4] V. F. Bardachenko, Analysis of the Characteristics of Time-Masked Information // USM. - 1994. - No. 3. - P.16-29.

[5] *N. Koblitz*, Algebraic aspects of cryptography. Vol. 3, Algorithms and Computation in Mathematics, Springer-Verlag, Berlin, 2004.– 207 p.

[6] Agnieska Danek. Application of the Burrows-Wheeler Transform for Searching for Approximate Tandem Repeats. Springer Springer-Verlag US, 2012, pp 256-256.

[7] Osadchyy Y.O., Osadchyy O.Y., Skuratovskii R.V. // Numerical regularities and timer coding information. Artificial intelligence. -№3.- 2017.-P.1-22.

[8] V.M. Tereshchenko, Y.O. Osadchyy, Pransforming technology of coding information in the computer of the von Neumann architecture. Research topics: International scientific youth school "Systems and means of artificial intelligence (AIIS`2017).- K.- 2017.- P.210-214.

[9] Turing A. M. On Computable Numbers, with an Application to the Entscheidungsproblem. A Correction // Proceedings of the London Mathematical Society — 1938. — Vol. s2-43, Iss. 6. — P. 544–546. — ISSN 0024-6115; 1460-244X — doi:10.1112/PLMS/S2-43.6.544

[10] Ruslan Skuratovskii. Parallel solution in fast methods of timer coding of information. High Performance Computing Kyiv, October 22-23, 2018. Source: http://hpc-ua.org/hpc-ua-18/participants/

[11] Bolotov A. A. Gashkov S. B., Frolov A. B., Chasovsky A. A. An Elementary Introduction to Elliptic Cryptography – CompBook Vol. 2 2006 – 328 p.

[12] Volkov Y. I., Voynalovich N. M. Elements of Discrete Mathematics. Kirovograd Central Ukrainian State Pedagogical University, 2000 y.– 174 p.

[13] Osadchyy Y.O., An Approach to Improvement of Timer Methods of Information Security // Vis. TANG, Economic and mathematical modeling. - 1999.– № 1.– P. 30–35 p.

[14] R. Skuratovskii, The Derived Subgroups of Sylow 2-Subgroups of the Alternating Group and Commutator Width of Wreath Product of Groups. Mathematics, Basel, Switzerland, (2020) № 8(4), pp. 1-19.

[15] Skuratovskii R. V., Osadchyy V. Order of Edwards and Elliptic Curves Over Finite Field. WSEAS Transactions on Mathematics, Volume 19, pp. 253-264, 2020.

[16] https://studfiles.net/preview/5368369/page:4/

[17] Gnatyuk, V. A. Mechanism of laser damage of transparent semiconductors.Physica B: Condensed Matter,. pp. 308-310, 2001

[18] Arthur Franz, Oleksandr Antonenko, Roman Soletskyi. A theory of incremental compression. Informatics and Computer Science Intelligent Systems Applications. 2020. Vol 540. pp 2-11.