Search for a substring of characters using the theory of non-deterministic finite automata and vector-character architecture

Dmitry V. Pashchenko[1], Dmitry A. Trokoz [2], Alexey I. Martyshkin[3], Mihail P. Sinev [4], Boris L. Svistunov[5]
[1]Penza State Technological University, Rusia
[2,3]Department of Computational Machines and Systems, Penza State Technological University, Rusia
[4]Department of Computer Science, Penza State University, Rusia
[5]Department of Mathematics and Physics, Penza State Technological University, Rusia

Article Info

Article history:

Received Aug 21, 2019
Revised Nov 12, 2019
Accepted Feb 22, 2020

Keywords:

Big data
Hyperdimensional vector
Information retrieval algorithms
Nondeterministic finite automata
Vector character architecture

ABSTRACT

The paper proposed an algorithm which purpose is searching for a substring of characters in a string. Principle of its operation is based on the theory of non-deterministic finite automata and vector-character architecture. It is able to provide the linear computational complexity of searching for a substring depending on the length of the searched string measured in the number of operations with hyperdimensional vectors when repeatedly searching for different strings in a target line. None of the existing algorithms has such a low level of computational complexity. The disadvantages of the proposed algorithm are the fact that the existing hardware implementations of computing systems for performing operations with hyperdimensional vectors require a large number of machine instructions, which reduces the gain from this algorithm. Despite this, in the future, it is possible to create a hardware implementation that can ensure the execution of operations with hyperdimensional vectors in one cycle, which will allow the proposed algorithm to be applied in practice.

Corresponding Author:

Dmitry A. Trokoz
Department of Computational Machines and Systems,
Penza State Technological University,
440039, Russia, Penza, 1/11 Baydukova proyezd/Gagarina ul, 1/11.
Email: dmitriy.trokoz@gmail.com

1. INTRODUCTION

Currently, there is a steady growth of information volume available for search and analysis, caused by continuous technological development in the field of information technology. The change in the nature and total volume of world data over the past few decades is described in [1-5] and is graphically presented in Figure 1. This phenomenon was called the term Big Data, which does not have a strict definition. However, in the literature, they usually imply a volume of data that cannot be completely processed by existing algorithms in a reasonable time [6] or in real time [7]. Undoubtedly, the search for information in a huge variety of data is one of the key problems for the solution of which new algorithms and methods are regularly proposed, including cloud computing [8] and specialized computing systems [9]. Since the nature of information is different, classical problems are distinguished, which underlie specialized search algorithms. One of these classic tasks is the task of searching for a substring of characters in a string. Many information retrieval tasks are reduced to this task, for example, information retrieval in database management systems and a number of other tasks. There are many well-known algorithms for solving this task, such as the Boyer-Moore-Horspul algorithm which uses the offset table 2, the Rabin-Karp algorithm based on hashing [10], the Knut-Morris-Pratt algorithm using the prefix function [11] and others. In the worst case, the best of these algorithms provide a linear dependence of computational complexity on the sum of the lengths of the target and desired strings. In this paper, we propose an algorithm that has a fixed linear complexity depending on the length of the desired string. It should be noted immediately that the computational complexity of the proposed algorithm is measured in vector operations. Therefore, the time complexity of the algorithm is determined by the availability of hardware support for such operations by the computing platform.

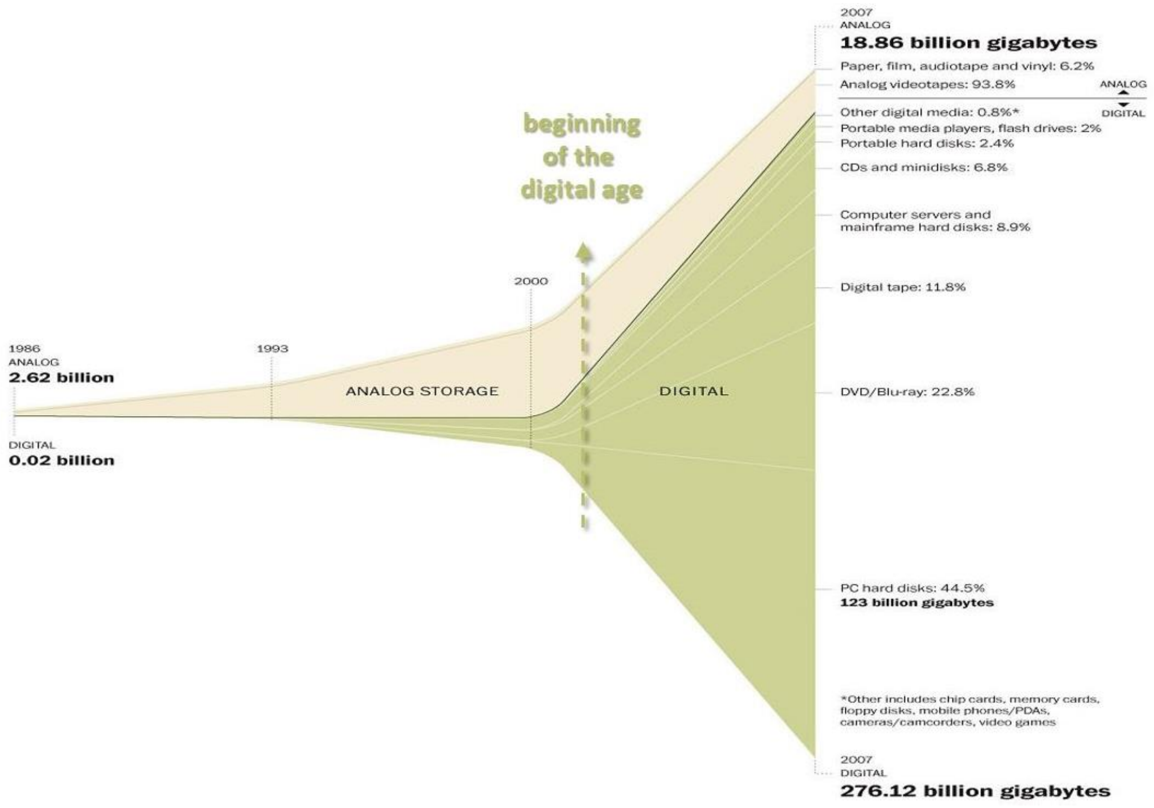

Figure 1. Changes in the nature and total volume of world data

The paper is structured as follows. Section 2 presents a small overview of the mathematical apparatus concerning non-deterministic finite automata; this overview is used later in section 3 for the formalized writing of a substring search algorithm for characters in a string that does not have linear dependence of complexity yet. Section 4 gives a brief description of the hyperdimensional vector algebra, which is the basis of the vector-character architecture and is used in section 5 for the synthesis of a hyperdimensional vector for an arbitrary non-deterministic automaton. Also in Section 5, it is shown that using the obtained hyperdimensional vector, it is possible for the number of operations, linearly dependent on the number of input characters, to determine the resulting state of the corresponding non-deterministic automaton, thereby ensuring the linear complexity of the algorithm described in Section 3. The output is presented in section 6.

2. RESEARCH METHOD

2.1. Nondeterministic finite state automata

An abstract automaton is a mathematical model of a discrete device that has one input, one output, and is at any time in one of the many internal states. An automaton is called finite if the set of its internal states is finite [12]. In this case, the total internal memory of the finite state automaton is equal to $\log_2 N$ bits, where N is the power of the set of states occupied by the finite automaton [12, 13]. There are several ways to specify a finite state automaton, we consider two of them, they will be used in this work.

The first method involves setting a finite state automaton in the form of a four-element tuple $<S, s_0, C, T>$, where;
S is a set of particular states of a finite automaton;
s_0 is an initial generalized state of an automaton;
C is a set of input characters of a finite state automaton;
T is a set of transitions, each element of which is a tuple of the following form: $t^j = < v^j, w^j, z^j >$, here v^j is the state from which the automaton passes under the influence of the signal w^j to z^j, i.e $v^j, z^j \in S$, $w^j \in C$.

The second way of setting is graphical, with the finite state automaton being represented as a labelled oriented graph, where the vertices are the states of the automaton and the arcs are transitions labelled with the characters of the input alphabet, under the action of which the corresponding transition occurs. In addition, the graph vertex is additionally labelled, usually with the character s_0; it is the initial state of the finite state automaton. An example of a graphical representation of a finite state automaton is shown in Figure 2.

The obvious disadvantage of classic finite automata (also called deterministic finite automata) is the small amount of data that the automaton is able to store as its internal state. The second disadvantage is the weak expressive abilities of deterministic finite automata expressed in the absence of means of describing parallel processes. Both of these problems solve the expansion of finite automata, called non-deterministic finite automata, which should not be confused with probabilistic finite automata.

A non-deterministic finite state automaton is a finite state automaton that allows for the availability of several transitions from one state to another under the action of the same input character, and non-deterministic automata also allow the possibility of using an empty character for the transition. As a consequence, a non-deterministic finite state automaton is capable of residing simultaneously in several states, called particular ones. At first glance, this contradicts the definition of an abstract automaton, which is a nondeterministic finite automaton. However, there is no contradiction here, since the complete state of the automaton is a subset of the particular states in which the automaton resides at the moment. Thus, the total number of complete states of an automaton is 2^N, where N is the number of particular states of a nondeterministic finite automaton. Thus, the number of complete states of a non-deterministic finite state automaton is much larger than the deterministic one; as a result, the amount of data that a non-deterministic finite state automaton is capable of storing is N bits, where N is the number of particular states of the automaton. This solves the first of the problems described above. The problem of simulating parallel processes arises from the definition of non-deterministic finite automata, namely, the ability to be simultaneously in several states, as a result, in one clock cycle of the automaton to move simultaneously, that is in parallel, into several other states.

An example of a non-deterministic finite state automaton is presented in Figure 3. It is easy to make sure that the machine is non-deterministic: it is enough to look at the transitions from the states s_0 and s_2, they are labelled with the same characters *(a* character labels transitions from the state s_0 and character *c* labels transitions from the state s_2).

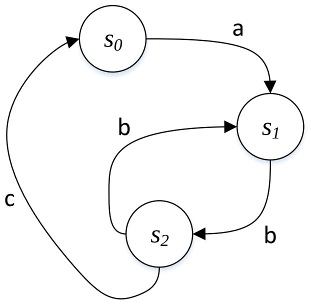

Figure 2. The finite state automaton, given in a graphical way

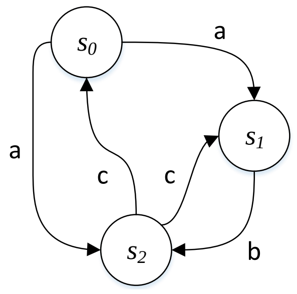

Figure 3. Example of a non-deterministic finite state automaton

However, another case is possible when an automaton is also non-deterministic, with no arc labelled with an empty character, and more than one arc is labelled with the same character doesn't go out of one state. Such a case is possible if the automaton initially exhibits the property of non-determinism, that is, its initial state is represented as a subset of several particular states. At the end of the consideration of the non-deterministic finite automata mathematical apparatus, we should mention the determinism theorem which states that for any non-deterministic finite automaton an equivalent deterministic finite automaton can be constructed. In this case, the number of states of an automaton after determinism in the general (worst) case is 2^N, where N is the number of particular states of a non-deterministic finite automaton.

3. RESULTS

As was shown above, the finite automata mathematical apparatus considered in the previous section allows describing parallel processes in contrast to deterministic finite automata. This property will be used in this section when describing the algorithm for solving the problem posed at the beginning of the work, namely, searching for a substring in the character string. The proposed algorithm allows determining only the fact of availability of the required string in the target string, but not its position. However, in the majority of cases, the latter is not required, for the overwhelming number of tasks using search for a substring, it is the answer to the question of availability that is important, without clarifying the position.

We take the word “hillbillies” as the target string, this word is interesting because it is the longest that you can “type” on the calculator. To do this, type 53177187714 and turn the calculator upside down. Of course, this is nothing more than a curious fact and the target word can be any, as well as the desired one, it will in no way affect the search result. We now proceed directly to the description of the search algorithm. It consists of the following:

a. We synthesize a nondeterministic finite state automaton for searching the necessary substrings.
b. We submit the next character of the string to the input of the automaton.
c. Go to the next character of the string, if it is not, go to step 5.
d. Go to step 2.
e. We check the complete state of the automaton (which is a subset of the particular states of the automaton) for emptiness. The power (potency) of this set is equal to the number of occurrences of the desired string in the target string. If the set is empty, then the substring is not found.

All points of the algorithm are simple and do not cause questions, except the first. We consider in detail how a non-deterministic automaton is synthesized to search for a substring. First of all, it should be noted that the structure of the automaton depends only on the target string and does not depend on the desired one; hence we can conclude that once synthesized, the automaton for the target string can be reused for the various required strings. The similar task (search of various substrings in the same strings e) meets rather often. It should be noted that the algorithm is effective in this case since the evaluation of the computational complexity of the algorithm does not read the time for the synthesis of the automaton, assuming that it will be synthesized once, while the search for substrings will be performed repeatedly for different substrings. All these limitations are important to consider when using the proposed algorithmic solution.

The synthesized automaton has a linear structure, the number of transitions is equal to the length of the target string, and the number of particular states is one more, respectively. The full initial state of the automaton is a subset that includes all particular states of the automaton (it is not necessary to include the last state, however; in Section 5 it will be shown that due to this greater unification is achieved and fewer operations are required). Let us describe the synthesis of the automaton in general form. Let the target string be given as an ordered set of characters $A=\{a_i\}$. Then a_1 is the first character of the string, a_2 is the second, and so on. The length of the string, respectively, is $n=|A|$. And the automaton can be represented as follows as shown in Figure 4.

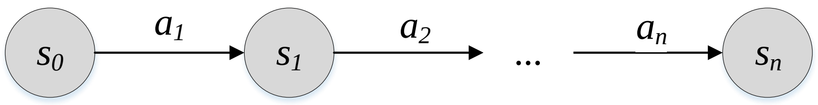

Figure 4. General view of a non-deterministic automaton for searching a substring

Here, grey color marks the particular states in which the automaton resides (as mentioned above, at the initial moment, these are all particular states). We go back to the example described above and implement the synthesis of a non-deterministic finite automaton to search for a substring in the target string “hillbillies”. The resulting automaton is shown in Figure 5.

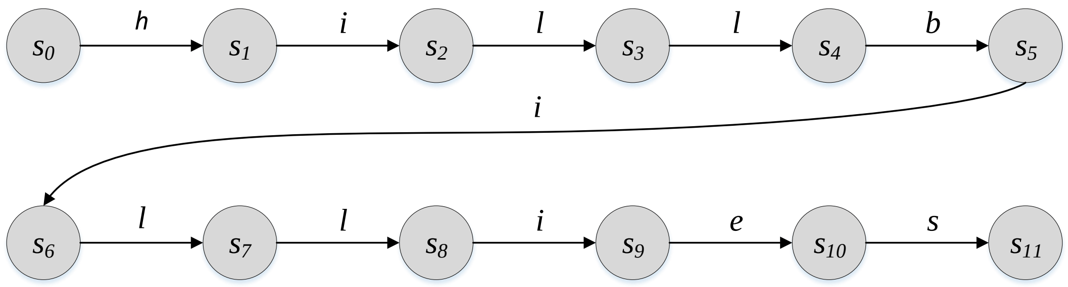

Figure 5. The automaton for the search "hillbillies" in the target string

We take “illb” as the desired substring; this substring is chosen for a more visual demonstration of the algorithm. We consider the algorithm step by step; at each step, we will submit one character of the desired string and see how the full state of the automaton changes (a subset of particular states). Particular states in which the automaton resides will also be labelled in grey. After submitting to the input of the automation of the character “i” which is the first character of the desired string, the automaton will go to the state shown in Figure 6.

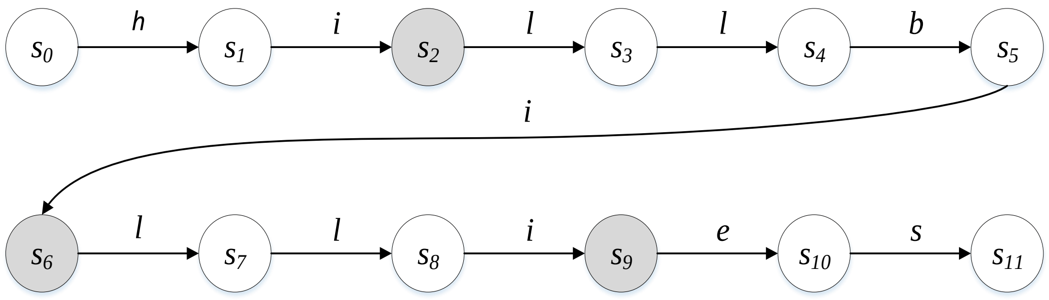

Figure 6. The automaton after delivery the character "i"

It can be seen from the above picture that the entered character “i”, or rather a substring consisting so far only of one character, occurs 3 times (this is equal to the power of the subset describing the full state of the automaton). Next, the second character of the required substring "l" is fed to the input of the automaton. The resulting state of the automaton is shown in Figure 7.

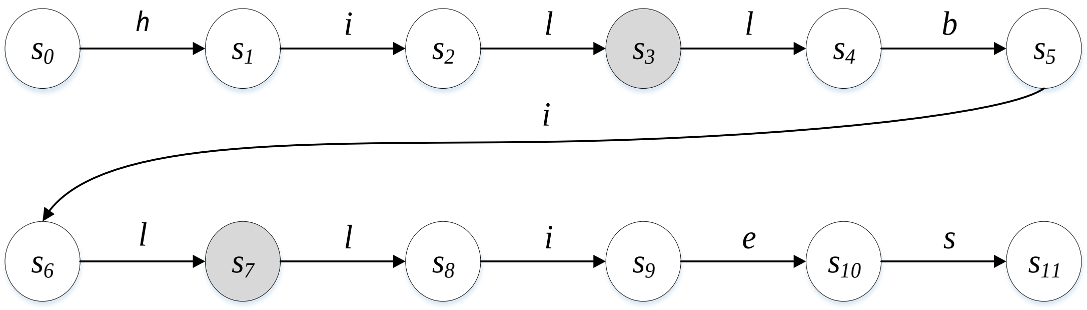

Figure 7. The automaton after entering the character sequence "il"

This figure shows that the sequence “il” occurs only twice (equal to the number of particular states in which the automaton resides). Then the third character of the required string "l" is delivered to the input. The resulting state of the automaton is shown in Figure 8.

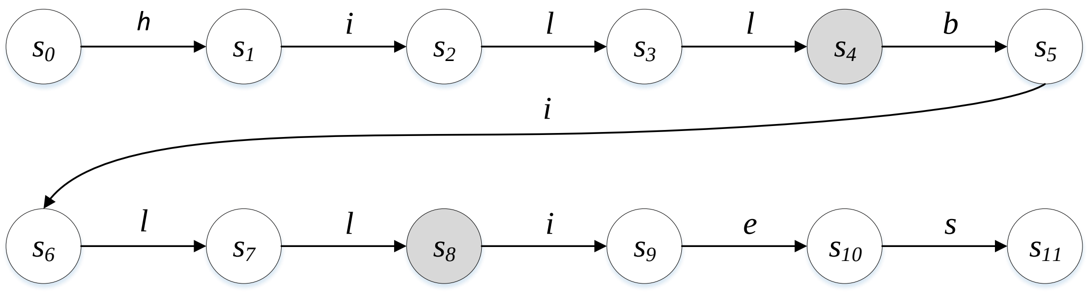

Figure 8. The automaton after entering the character sequence “ill”

At the last step, the character “b” is fed to the input of the automaton, which is the last character of the desired substring. The resulting state of the automaton is shown in Figure 9. Figure 9 shows that after delivery of the desired substring to the input of the automaton, the last will go to the full state, including only one particular state; it follows that the required substring was found in the target string, and only once. The described algorithm in its present form has a computational complexity equal to the product of the lengths of the target and the desired strings, because at each step, in general, all transitions of the automaton (the number of which, as was shown earlier, is equal to the length of the target string) can actuate, and the number of steps is equal to the length of the required strings. The following sections briefly describe the vector-character architecture and propose on its basis a simulation method for a non-deterministic automaton that, within this architecture, requires a number of operations which is linearly dependent on the length of the input sequence. This will allow the achievement of the linear complexity of the described search algorithm.

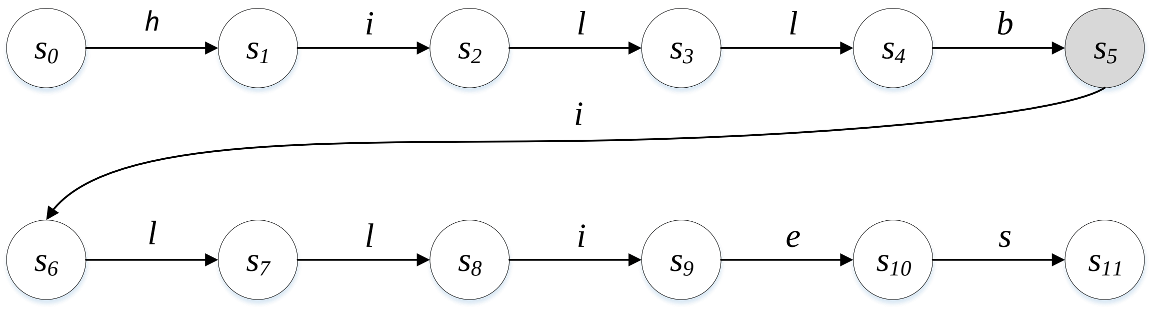

Figure 9. The resultant state of the automaton after the required substring “illb” is sent to the input

3.1. Vector-character architecture

Vector-character architecture is a well-established name for the presentation of semantically related information. It is based on bio-inspired methods which are based on one of the mechanisms of brain activity. As it is known, relatively simple mental events involve a very large number of scattered neurons, similarly, the vector-character architecture uses a distributed representation of information, so that one logical entity is associated with a large number of codes [14-20]. There are many different types of vector-character architecture using different representations, for example, [12-14]. In this paper, the vector-character architecture based on the so-called binary sprinkling codes [15] is used; it applies binary vectors and an “exclusive or” operation to multiply them.

Cognitive abilities achieved using vector-character architectures were demonstrated in works [16, 17] devoted to imitating learning systems of honey bees. In addition, approaches to the use of vector-character architectures for solving progressive Raven matrices [18, 19, 21] were proposed. The use of vector-character architectures of various classes for searching textual information has been proposed in a number of papers [20-22]. The methods discussed in these papers use sequence analysis and a vector-character architecture for the distributed presentation of analysis results and subsequent comparison. The use of similar methods to search for more general patterns, including graphical ones, was considered in works [23, 24].

The paper proposes to combine the advantages of vector-character architecture and the mathematical apparatus of the non-deterministic finite automata theory to search for a substring in a string. The advantage of the proposed method is to reduce the required number of operations with binary vectors, which depends only on the length of the desired string, but not on the length of the string that is being searched. The method is based on the algorithm for converting a non-deterministic finite state automaton obtained using the approach described earlier into a vector-character form, which allows for a given input string of characters to get the resulting state of the automaton for the number of vector operations linearly dependent on the number of characters in the input string. One more additional vector operation will be needed to analyze the resultant state of the automaton, which will not have a significant impact on the number of operations and the linear nature of the dependence of their number on the length of the required string.

Before proceeding directly to the algorithm for converting a non-deterministic finite automaton into a vector-character form, we briefly consider the main vector operations and their properties that will be needed to understand the proposed algorithm. A hyperdimensional vector is a binary vector of high dimensionality that contains a random sequence of zeros and ones obtained using the uniform distribution law [25]. The primary metric which is used when comparing two hyperdimensional vectors is Hamming distance. Hamming distance is absolute (measured in bits) and relative (measured in fractions). In this paper, we use the relative Hamming distance d_h, which for two vectors "a" and "b" of length L=|a|=|b| is equal to the fraction of the bitwise non-coincidental decades in these vectors:

$$d_h(a,b) = \frac{\sum_{i=1}^{N} a_i \oplus b_i}{L} \tag{1}$$

In [25], it was shown that the Hamming distance between two random hyperdimensional vectors is close to 0.5. This is an important property that will later be used in analyzing the results of algebraic operations with hyperdimensional vectors.

3.2. Sum of hyperdimensional vectors

The sum of n hyperdimensional vectors is a hyperdimensional vector such that each of its bits is equal to the dominant value of the bit in the corresponding decade. Thus, if the sum of bits in a decade is greater than n/2, then the resulting bit is 1, and if less, then 0. In this case, an odd number of vectors must participate in the sum [26]. The last restriction can be removed by introducing the following rule: if the number of one and zero bits in the corresponding decade of summable vectors is the same, then the resulting bit is filled with a random value. A hyperdimensional vector obtained as a result of summation has the following properties [22]:

- The number and order of ones and zeros in the composition of the vector obeys the uniform distribution law.
- The resulting vector is "similar" to all the vectors included in the sum.
- If the sum includes several copies of one of the vectors, then the resulting vector is closer (according to Hamming) to this vector than to any other.

The second property is the most important among the listed properties of the hyperdimensional vector sum. The term "similar" is very vague, so we'll clarify that Hamming proximity, which is different from the mean, is understood here, which, as mentioned above, is approximately equal to 0.5 for any two random hyperdimensional vectors. In the works [11] and [13], it is proposed to assume that the Hamming distance between any sum vector and the resulting vector should not exceed 0.47. However, this value is empirical by its nature; it is obtained for vectors of dimension 10000, for vectors of higher dimension it can be more than 0.47, and less than 0.47 for vectors of smaller dimension [11]. Within the framework of solving the problem, it is important with a high degree of probability to ensure the possibility of separating the hyperdimensional vectors included in the sum and random hyperdimensional vectors that are not members of the sum. In general, we follow the rule: the more vectors in the sum, the greater the Hamming distance, and the higher the dimension of the vector, the smaller it is. Formally, this operation can be represented as follows for the set $A=\{a^j\}$ of hyperdimensional vectors of length l, where s is a resultant vector, and *RND* is a function that returns a random binary value:

$$s_i(A) = \begin{cases} \left[\dfrac{\sum_{j=1}^{|A|} a_i^j}{l} \right], \dfrac{\sum_{j=1}^{|A|} a_i^j}{l} \neq 0,5 \\ RND(), \dfrac{\sum_{j=1}^{|A|} a_i^j}{l} = 0,5 \end{cases} \tag{2}$$

The resulting vector is actually a subset of the vectors included in the sum. Knowing the resulting vector and the total set of all hyperdimensional vectors used in the task, we can determine the vectors included in the sum. To better understand what it is all about, we consider an example. Let the solution of the task use the set of hyperdimensional vectors *{a, b, c}*. In the course of the calculations, a vector was obtained, which is the sum of s vectors of the subset of the given set. Then, using the second property of the addition operation and the operation of calculating the Hamming distance between the vectors discussed above, we can determine the summable subset of vectors (vectors included in the sum). For this, we will consistently find the Hamming distance between the resultant sum vector and each vector of the set *{a, b, c}*. Suppose the result is the following:

d_h (s, a)=0.35
d_h (s, b)=0.38
d_h (s, c)=0.49

After analyzing the above data, taking into account the second property of the sum of vectors, we can conclude that the subset of summable vectors involved vectors a and b, but not vector c, that is, s=a+b. We can conclude that the resulting sum vector of hyperdimensional vectors can be decomposed into summands, that is, the sum operation for hyperdimensional vectors is reversible.

3.3. Multiplication of hyperdimensional vectors

The product of two hyperdimensional vectors *a* and *b* is such a hyperdimensional vector *c*, each of the bits of which is defined as the "exclusive or" values of the bits of the vectors *a* and *b* in the corresponding bit. The hyperdimensional vector obtained as a result of multiplication has the following properties:

- The number and order of ones and zeros in the composition of the vector obeys the law of uniform distribution.
- The resulting vector "does not look like" the vectors involved in the product.

The last property says that the Hamming distance between the resulting vector of the product and any factor is approximately 0.5. Formally, the multiplication operation of a subset of hyperdimensional vectors *A={a ^{j}}*, as a result of which the vector p is obtained, can be represented as follows:

$$p_i = \bigoplus_{j=1}^{|A|} a_i^j \tag{3}$$

In addition, two important properties of the multiplication operation can be distinguished:

- The multiplication of hyperdimensional vectors is distributive with respect to addition.
- Multiplying hyperdimensional vectors is the inverse operation for itself.

The second property is characteristic of the “exclusive or” operation, which underlies the multiplication operation for hyperdimensional vectors. This property is usually used when working with hyperdimensional vectors to extract values in structured data represented as a hyperdimensional vector. For a better understanding, consider an example. Suppose we have a structure containing 3 fields *a*, *b*, *c*, each of which can contain one of the values of the subset *V={v ^{j}}*. Suppose that a field, *a, contains v ^{2}*, *b-v ^{6}*, *c-v ^{3}*. In addition, we represent each of the fields and values of the subset *V* in the form of a hyperdimensional vector. Then we can write this structure with the corresponding values in the form of the following vector: $r = a \times v^2 + b \times v^6 + c \times v^3$, that is, a vector equal to the sum of pairwise products of hyperdimensional vectors of fields and values. Now, to obtain the value of the advancing field from the result vector, it is enough to multiply the resultant vector by the vector of the desired field. Let it be necessary to obtain the value of the field *b*, for this we multiply the resultant vector by the vector *b*, obtaining the vector *x*:

$$x = r \times b = (a \times v^2 + b \times v^6 + c \times v^3) \times b = a \times v^2 \times b + v^6 + c \times v^3 \times b \tag{4}$$

Using the second property of the vector product and the reversibility of the addition operation, we can determine that the vector v^6 is included in the resulting sum. To do this, we find the Hamming distance from the vector *x* to each vector of the subset *V*. For all other vectors, the Hamming distance will be approximately 0.5 and only for the vector v^6, it will be smaller. Thus, the multiplication operation can be used for storing in multidimensional information of multi-class information and its subsequent isolation.

3.4. Cyclic shift as a special case of bit permutation

The bit permutation operation is the third of the basic operations in the hyperdimensional vector algebra. In general terms, it can be represented as the product of a vector and a permutation matrix. It is also obvious that the total number of different permutations equals the factorial of the vector length.

In practice, a special case of a permutation of bits is often used, namely, a cyclic shift of the vector to the left or a cyclic shift of the vector to the right. These operations are inverse to each other and much simpler and shorter in writing than a permutation since the amount of shift is given by a natural number, while a matrix is needed for the permutation of bits. The cyclic shift operation for several bits can always be replaced by several successive cyclic shift operations by one bit. This operation is referred to as unary ones since only one hyperdimensional vector participates in it. In this case, the properties of cyclic shift operations coincide with the properties of the bit permutation operation. The hyperdimensional vector obtained as a result of the shift has the following properties:

- The number and order of ones and zeros in the composition of the vector obeys the uniform distribution law.
- The resulting vector "does not look like" the original vector, except for the case when the shift is made by a number multiple of the vector length.

Formally, the operation of a cyclic shift to the right *shr* of the hyperdimensional vector *a of* length *l* by *n* digits, as a result of which the vector *r* is obtained, can be represented as follows when numbering the bits from zero:

$$r_i = a_{(i+n)\bmod l} \tag{5}$$

and the operation of a cyclic shift to the left *shl* in the following form:

$$r_i = a_{(i-n+\left\lceil \frac{n}{l} \right\rceil * l)\bmod l} \tag{6}$$

Since the left and right cyclic shift operations are inverse to each other, the following property is obvious for any hyperdimensional vector *a* and a natural number *n*:

$$shr(shl(a,n),n) = shl(shr(a,n),n) = a \tag{7}$$

This property of the cyclic shift operation, as well as other properties of operations with hyperdimensional vectors, will be used in the next section when describing the algorithm for synthesizing a hyperdimensional vector for a given non-deterministic finite automaton.

3.5. Synthesis of a hyperdimensional vector describing a given non-deterministic finite state automaton

In the previous section, we considered the vector-character architecture and algebra for hyperdimensional vectors, which underlies it. This is necessary for understanding this section of the work, which is devoted to the method of representing a non-deterministic finite automaton as a hyperdimensional vector that will allow us to obtain the linear complexity of calculating the finite subset of states into which the automaton will go for a given string of input characters.

To represent a non-deterministic finite automaton in the form of a hyperdimensional vector, we extend the analogous method for a deterministic finite automaton proposed in the work [25]. Consider the hyperdimensional vector synthesis process for an arbitrary non-deterministic finite automaton which is defined by the following tuple: $<S, S_0, C, T>$, where *S* is the set of particular states of a non-deterministic finite automaton;
S_0 is a subset of particular states of a non-deterministic finite automaton that defines the initial generalized state of the automaton;
C is the set of input characters of a non-deterministic finite state automaton;

T - a set of transitions, each element of which is a tuple of the following form $t^j = < v^j, w^j, z^j >$, here v^j is a particular state from which the automaton goes under the action of the signal w^j to particular state z^j, i.e $v^j, z^j \in S$, $w^j \in C$.

If we assign a random hyperdimensional vector to each particular state and the input character of the automaton, then the hyperdimensional vector *A*, which defines the nondeterministic finite automaton, can be formally defined as follows:

$$A = \mathop{+}_{j=1}^{|T|} v^j \times w^j \times shl(z^j, 1) \tag{8}$$

In this case, the hyperdimensional vector S $_i$ defining the *i* -th generalized state of a non-deterministic finite automaton can be represented as a sum of hyper-dimensional vectors corresponding to particular states:

$$S_i = \mathop{+}_{j=1}^{|S_i|} s_i^j \tag{9}$$

To obtain a hyperdimensional vector of the following generalized state of a non-deterministic finite state automaton, it is necessary to perform a cyclic right shift for the product of the hyper-dimensional vector of the previous generalized state of a non-deterministic finite state automaton S_{prev}, a vector defining a non-deterministic finite automaton *A*, and an input character vector $c \in C$, under the action of which the transition occurs:

$$S_{next} \approx shr(S_{prev} \times A \times c, 1) \tag{10}$$

Here, an approximation sign is used, since in reality, a sum containing, in addition to the vectors of particular states of the automaton, forming the vector of the next generalized state of a non-deterministic finite automaton, also a number of vectors representing noise (vectors unlike any of the vectors of particular states and input signals of the automaton).

In order to better understand how a vector of the next generalized state of the automaton forms, we consider an example of the automaton for the string "hello", which allows us to solve the search task in this string of an arbitrary substring. At the same time, the computational complexity of solving this task measured in the number of operations with hyperdimensional vectors, will linearly depend on the length of the desired string (provided that the automaton is either already synthesized, or the search is performed many times, and therefore the synthesis process of the automaton performed once does not make significant contribution to algorithmic complexity). A nondeterministic finite state automaton for searching a substring in the target string "hello" is presented in Figure 10.

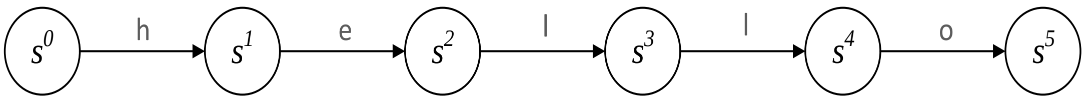
$$\begin{tikzcd} s^0 \arrow[r, "h"] & s^1 \arrow[r, "e"] & s^2 \arrow[r, "l"] & s^3 \arrow[r, "l"] & s^4 \arrow[r, "o"] & s^5 \end{tikzcd}$$

Figure 10. Nondeterministic automaton for searching a substring in the target string "hello"

Its generalized initial state $S_0 = S_{full} = \{s^0, s^1, s^2, s^3, s^4, s^5\}$ will include all particular states of the automaton, which later will also be used when checking its resulting state. As was shown earlier, if after the sought-for substring was delivered to the input of the automaton, the resulting generalized state is a non-empty subset of the particular states of the automaton, and the substring is found, otherwise it is not found. Synthesize the hyperdimensional vector for the given automaton:

$$A = s^0 \times h \times shl(s^1,1) + s^1 \times e \times shl(s^2,1) + s^2 \times l \times shl(s^3,1) + s^3 \times l \times shl(s^4,1) + s^4 \times o \times shl(s^5,1) \tag{11}$$

Now we search for the substring "ell" in the specified string. To do this, we will sequentially perform the operation of forming a new state for each of the characters of the desired string in accordance with expression (10):

At the input, "e" is delivered:

$$\begin{aligned}&S_e \approx shr(S_0 \times A \times e,1) = shr((s^0 + s^1 + s^2 + s^3 + s^4 + s^5)\times\\&\times(s^0 \times h \times shl(s^1,1) + s^1 \times e \times shl(s^2,1) + s^2 \times l \times shl(s^3,1) + s^3 \times l \times shl(s^4,1) + s^4 \times o \times shl(s^5,1)) \times e,1) =\\&= s^2 + noise\end{aligned}$$

At the input, "el" is delivered:

$$\begin{aligned}&S_{el} \approx shr(S_e \times A \times e,1) = shr((s^2 + noise)\times\\&\times(s^0 \times h \times shl(s^1,1) + s^1 \times e \times shl(s^2,1) + s^2 \times l \times shl(s^3,1) + s^3 \times l \times shl(s^4,1) + s^4 \times o \times shl(s^5,1)) \times l,1) =\\&= s^3 + noise\end{aligned}$$

At the input, "ell" is delivered:

$$\begin{aligned}&S_{ell} \approx shr(S_{el} \times A \times e,1) = shr((s^3 + noise)\times\\&\times(s^0 \times h \times shl(s^1,1) + s^1 \times e \times shl(s^2,1) + s^2 \times l \times shl(s^3,1) + s^3 \times l \times shl(s^4,1) + s^4 \times o \times shl(s^5,1)) \times l,1) =\\&= s^4 + noise\end{aligned}$$

In the above expressions, *noise* is noise which is the sum of vectors that are not similar to any of the vectors of particular states or input signals of the automaton. To check that the elements of the resulting subset representing the resulting state of the automaton are present, it is sufficient to find the Hamming distance between it and the full vector of particular states of the automaton. If $d_h (S_{ell}, S_{full}) < 0.47$, then the required string is found, otherwise - no. As it was stated earlier, 0.47 is an empirically determined value which is given as an example. The actual value should be determined within the framework of a specific task. Obviously, the longer the string being searched for and the more complex the automaton is (as a result of that the string where the search will be performed will be longer), the more noise will occur and the higher dimension vectors should be used.

When using the proposed method, the number of operations required to search for each of the characters of the desired string will be 3 (two multiplication operations and one cyclic shift). In addition, in the end, we will need to perform one operation to find the Hamming distance between the resultant vector and the full vector of particular states. Thus, the total number of operations required to search for a string of length M will be defined as 3 * M + 1, that is, the complexity will be linear.

Despite the linear dependence of the search algorithm complexity on the length of the desired string, which is characteristic of the method proposed in the work with the use of modern computing systems, it is unlikely to be able to achieve acceleration. This is due to the fact that operations with high-dimensional vectors performed on modern computers are resource-intensive and are performed far from within one automaton instruction. However, in the future, if the proposed architecture gets a hardware implementation, it will be possible thanks to the proposed method to ensure the linear complexity of finding a substring in a string depending on the length of the desired string not only in theoretical but also in practical terms.

4. CONCLUSION

The paper proposed an algorithm for searching substrings of characters in a string, based on the non-deterministic automata theory and vector-character architecture. The algorithm is able to provide a linear dependence of the search complexity on the length of the desired string, measured in the number of operations with hyperdimensional vectors. None of the existing algorithms has such a low level of computational complexity. However, hardware implementations of existing computing systems to perform operations with hyperdimensional vectors require a large number of automaton instructions, which reduces the gain from the proposed algorithm. Despite this, in the future, it is possible to create a hardware implementation that can ensure the execution of operations with hyperdimensional vectors in one cycle, which will allow the proposed algorithm to be applied in practice. Since this task is a key one in big data analysis systems, as well as when searching for information in databases, the proposed algorithm will provide a significant increase in speed in terms of searching for a wide class of information and computing systems.

ACKNOWLEDGEMENTS

Research was supported by the RFBR Grant for Best Projects of Basic Research, Grant No. 19-07-00516 A.

REFERENCES

[1] M. Hilbert and P. López, "The world's technological capacity to store, communicate, and compute information," *Science*, vol. 332, no. 6025, pp. 60-65, Apr 2011.

[2] H. Chen, et al., "Business intelligence and analytics: From big data to big impact," *MIS quarterly*, vol. 36, no. 4, pp. 1165-1188, Dec 2012.

[3] A. McAfee and E. Brynjolfsson., "Big data: the management revolution," *Harvard business review*, vol. 90, no. 10, pp. 60-68, Oct 2012.

[4] D. V. Paschchenko, et al., "Directly executable formal models of middleware for MANET and Cloud Networking and Computing," *Journal of Physics: Conference Series*, vol. 710, pp. 12024-12035, 2016.

[5] A. I. Martyshkin, et al., "Associative Co-processor on the Basis of Programmable Logical Integrated Circuits for Special Purpose Computer Systems," *Proceedings - 2018 Global Smart Industry Conference*, Nov 2018.

[6] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," *Communications of the ACM*, vol. 20, no. 10, pp. 762-772, 1977.

[7] R. M. Karp and M. O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM Journal of Research and Development,* vol. 31, no. 2, pp. 249-260, Mar 1987.

[8] D. E. Knuth, et al., "Fast pattern matching in strings," *SIAM Journal on Computing,* vol. 6, no. 2, pp. 323-350, 1977.

[9] B. A. Trakhtenbrot, "Finite automata. Behavior and synthesis (Fundamental Studies in Computer Science)," *American Elsevier*, 1973.

[10] A. W. Biermann and J. A. Feldman, "On the synthesis of finite-state machines from samples of their behaviour," *IEEE Transactions on Computers*, vol. C-21, no. 6, pp. 592-597, Jun 1972.

[11] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation,* vol. 1, pp. 139-159, Jan 2009.

[12] D. A. Rachkovskij and E. Kussul, "Binding and normalization of binary sparse distributed representations by context-dependent thinning," *Neural Computation,* vol. 13, no. 2, pp. 411-452, Feb 2001.

[13] D. Aerts, et al., "Geometric analogue of holographic reduced representation," *Journal of Mathematical Psychology,* vol. 53, no. 5, pp. 389-398, Oct 2009.

[14] R. W. Gayler, "Multiplicative binding, representation operators & analogy," *Advances in analogy research: Integration of theory and data from the cognitive, computational, and neural sciences,* pp. 1-4, Jan 1998.

[15] P. Kanerva, "Fully distributed representation," in *Proceeding of 1997 Real World Computing Symposium*, pp. 358-365, Jan 1997.

[16] D. Kleyko, et al., "Imitation of honey bees' concept learning processes using vector symbolic architectures," *Biologically Inspired Cognitive Architectures,* vol. 14, pp. 57-72, Oct 2015.

[17] D. Kleyko, et al., "Fly-the-Bee: A game imitating concept learning in bees," *Procedia Computer Sciences*, vol. 71, pp. 25-30, Dec 2015.

[18] B. Emruli, et al., "Analogical mapping and inference with binary spatter codes and sparse distributed memory," in *Proceeding International Joint Conference on Neural Networks (IJCNN)*, pp. 1-8, 2013.

[19] D. Rasmussen and C. Eliasmith, "A neural model of rule generation in inductive reasoning," *Topics in Cognitive Sciences,* vol. 3, no. 1, pp. 140-153, Jan 2011.

[20] I. Misuno, et al., "Searching for text information with vector representations," *Journal of Problems in Programming*, vol. 4, pp. 50-59, 2005.

[21] A. Sokolov, "Vector representations for efficient comparison and search for similar strings," *Cybernetics and Systems Analysis,* vol. 43, no. 4, pp. 484-498, Jul 2007.

[22] D. Kleyko and E. Osipov, "On Bidirectional Transitions between Localist and Distributed Representations: The Case of Common Substrings Search Using Vector Symbolic Architecture," *Procedia Computer Science*, vol. 41, pp. 104-113, 2014.

[23] D. Kleyko and E. Osipov, "Brain-like classier of temporal patterns," in *The Proceeding of the 2nd International Conference on Computer and Information Sciences - ICCOINS*, pp. 1-6, 2014.

[24] D. Kleyko, et al., "Holographic Graph Neuron: A Bioinspired Architecture for Pattern Processing," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 6, pp. 1250-1262, Jun 2017.

[25] E. Osipov, et al., "Associative synthesis of finite state automata model of a controlled object with hyperdimensional computing," in *Proceedings IECON 2017: 43rd Annual Conference of the IEEE Industrial Electronics Society*, pp. 3276-3281, 2017.

[26] M. Ganjali and B. Teimourpour, "Identify Valuable Customers of Taavon Insurance in Field of Life Insurance with Data Mining Approach," *UCT Journal of Research in Science, Engineering and Technology*, vol. 4, no. 1, pp. 1-10, 2016.

BIOGRAPHIES OF AUTHORS

Dmitry Vladimirovich Pashchenko in 1998 received a diploma in "Computing machines, complexes, systems and networks". In 2003, he defended his PhD thesis on specialties "System analysis management and information processing" and "Mathematical and software of computers, complexes and computer networks". In 2013-defended his doctoral dissertation on specialty "System analysis management and information processing". Since September 2019-rector of the Penza state technological University. The main areas of activity are: modeling and formalization of models of multi-threaded computing systems using the mathematical apparatus of Petri nets; development of parallel algorithms for expert systems; hardware and software implementation of high-performance computing systems. E-mail: dmitry.pashcenko@gmail.com

Dmitry Anatolyevich Trokoz in 2011 received a master's degree, in 2013 a Ph.D. in technical sciences in the field of theoretical foundations of computer science in Russia, at Penza State University. Since January 2014, he worked as a teacher at PSU as an assistant professor at the Department of Computing Engineering, and since September 2019, he has been working as a teacher at Penza State Technological University. The main areas of activity are: modeling and formalization of models of multi-threaded computing systems using the mathematical apparatus of Petri nets; development of parallel algorithms for expert systems; hardware and software implementation of high-performance computing systems. E-mail: dmitriy.trokoz@gmail.com

Alexey Ivanovich Martyshkin in 2010 received a diploma in "Computing machines, complexes, systems and networks", in 2013 a Ph.D. in technical sciences in the field of mathematical modeling, numerical methods and software complexes in Russia, at Penza State Technological University (PSTU). Since September 2011, he worked as a teacher at PSTU as an assistant and since 2014 he has been working as an assistant professor at the Department of Computing Systems. The main areas of activity are: modeling of models of multi-threaded and multiprocessors computing systems using the mathematical apparatus of systems and Queuing networks; development of parallel algorithms for expert systems; hardware and software implementation of high-performance computing systems. E-mail: alexey314@yandex.ru

Sinev Mikhail Petrovich in 2011 received a master's degree, in 2014 received a Ph.D. in technical sciences in Russia, from Penza State University in the field of theoretical foundations of computer science. Since 2014, he has been working as a teacher at PSU as an assistant professor at the Department of Computer Engineering. The main field of activity are: development of parallel algorithms for expert systems; hardware and software implementation of high-performance computing systems. E-mail: mix.sinev@gmail.com

Boris Lvovich Svistunov in 1971 received a diploma in "Automation and telemechanics", in 2004 received a doctoral thesis in the field of devices and measurement methods in Russia, at the Penza state University. Works as a Professor of the Department of Mathematics and physics. The main activities are: the use of structural and algorithmic redundancy in devices for measuring and monitoring parameters of electric circuits and output values of parametric sensors E-mail: sbl@penzgtu.ru