DOI: http://doi.org/10.25728/cs.2021.4.1

# NATURAL COMPUTING WITH APPLICATION TO RISK MANAGEMENT IN COMPLEX SYSTEMS

A.A. Shiroky and A.O. Kalashnikov

Trapeznikov Institute of Control Sciences, Russian Academy of Sciences, Moscow, Russia

✉ shiroky@ipu.ru, ✉ aokalash@ipu.ru

**Abstract.** This paper surveys natural computing models and methods with application to risk management in complex systems. The equivalence of risk minimization and effective control problems is shown. The general risk management problem is stated for complex systems under uncertainty. The structure of fundamental and applied risk management problems is described. The well-known natural computing methods are briefly considered with application to risk management by the criteria of formalism, universality, and learning capability. The scientific community's preferences in natural computing models and methods for solving different classes of risk management problems are analyzed. Some promising approaches are outlined, which are currently underinvestigated according to the authors' opinion.

**Keywords:** risk management, effective control problem, natural computing.

## INTRODUCTION

Many control problems for complex systems can be reduced to assessing and minimizing risks of the controlled system or interpreted in these terms. Risk management is an integral problem involving subproblems, including identification, monitoring, forecasting, optimization, etc. Effective solutions are impossible under continuously circulating volumetric data flows without building sufficiently complex models of controlled systems and applying adequate computational methods. The real dynamic systems for which such problems are considered are open. Hence, the methods used should adapt the models to a continuously changing (particularly unpredictable) environment.

In this regard, an analogy with the processes in living nature suggests itself: living organisms permanently struggle for survival in a variable environment causing both predictable and unpredictable threats. Analysis of the behavioral models of living organisms, group control structures, and protection mechanisms against external threats seems to be very promising due to the proven operability of such models (there are many examples in nature) and their high effectiveness achieved in the course of evolutionary processes. At the moment, natural computing is an actively developing field of science that combines element connectivity, group behavior, and emergence. In the book [1], the term "*natural computing*" was defined as an area of research that studies models and computational methods inspired by nature and considers the phenomena occurring in nature from the information processing viewpoint.

Many branches of natural computing (e.g., artificial neural networks) are now at their peak of popularity; others (e.g., bacterial and DNA computing) are relatively little investigated. This paper surveys the main known classes of natural computing algorithms and their application to risk management. Note that describing the approaches themselves is not the goal of the authors.

Further presentation is organized as follows. In Section 1, the general risk management problem is stated for complex systems under uncertainty, and its equivalence to the effective control problem is shown. Section 2 considers fundamental and applied risk management problems for complex systems, identifying those solved by natural computing methods. In Section 3, we suggest possible classification bases of natural computing, highlighting the ones important for applying the described algorithms and methods to risk

management in complex systems. Section 4 overviews well-known natural computing methods and algorithms and classifies them by the bases selected in Section 3. Finally, Section 5 outlines some promising approaches currently underinvestigated according to the authors' opinion.

## 1. RISK MANAGEMENT IN COMPLEX SYSTEMS UNDER UNCERTAINTY: GENERAL PROBLEM STATEMENT

In this paper, complex systems are systems with an infinite variety of responses to external actions. In the most general statement, the control problem is to find control actions transferring a complex system to a target state. Such control actions are called *effective*. Formally, this problem can be written as follows.

Let $U$ be the set of control actions, and let $Q$ and $\Theta$ be the sets of system states and environment's states, respectively. Assume that the system state $q \in Q$ is uniquely determined by the environment's state and control actions applied: $q = q(u, \theta)$, where $u \in U$ and $\theta \in \Theta$. We introduce a functional $K = K(u, q, \theta) = K(q(u, \theta))$, further referred to as *the optimality criterion*, which depends on the control action $u$, the system state $q$, and the environment's state $\theta$. The problem is finding a *globally optimal* control action $u^* \in U$ such that

$$K(u^*, q, \theta) = \max_{u \in U} K(u, q, \theta). \quad (1)$$

In the degenerate case, when the controller "knows" the actual states of the controlled system $q_0 \in Q$ and environment $\theta_0 \in \Theta$, the optimality criterion is a function of one variable, and problem (1) reduces to:

$$K(u^*, q_0, \theta_0) = \max_{u \in U} K(u, q_0, \theta_0). \quad (2)$$

Suppose that there exists a set of (nonunique) variables $(u^*, q^*, \theta^*)$ for which

$$K(u^*, q^*, \theta^*) = \max_{u \in U} \max_{q \in Q} \max_{\theta \in \Theta} K(u, q, \theta)$$

(*the perfect statement*).

The function

$$\varphi(u, q, \theta) = K(u^*, q^*, \theta^*) - K(u, q, \theta),$$

representing the difference between the maximum and current values of the optimality criterion (for some $u$, $q$, and $\theta$), will be called *the loss function*. Clearly,

$$\min_{u \in U} \varphi(u, q, \theta) = K(u^*, q^*, \theta^*) - \max_{u \in U} K(u, q, \theta).$$

In other words, maximization of the optimality criterion is equivalent to minimization of the loss function. In the deterministic case, the equivalence holds as well:

$$\min_{u \in U} \varphi(u, q_0, \theta_0) = K(u^*, q^*, \theta^*) - \max_{u \in U} K(u, q_0, \theta_0). \quad (3)$$

In real problem statements, complete awareness is rare, mainly occurring in technical systems control. As a rule, the controller has incomplete and (or) unreliable information about the states of the controlled system and environment. Hence, to solve the problem, the uncertainty must be eliminated somehow. For example, it is possible to choose the most probable or least favorable states, or act differently; see the paper [2].

Let $u \in U$ be a fixed control action. We define an uncertainty elimination operator $\mathfrak{I}$: the result of applying it to the optimality criterion will depend on the chosen control action $u$ only:

$$\mathfrak{K}(u) = \mathfrak{I}K(u, \cdot, \cdot).$$

By analogy with problem (2), we find a globally optimal control action $u^*$ that will maximize the optimality criterion under uncertainty:

$$\mathfrak{K}(u^*) = \max_{u \in U} \mathfrak{K}(u). \quad (4)$$

We introduce a real-valued *risk function*

$$\rho(u) = \mathfrak{K}(u^*) - \mathfrak{K}(u).$$

Generally, *risk* is often defined as a system parameter and a property of a controller (particularly, a decision-maker) to decide under uncertainty, which can cause both undesirable (dangerous) and significantly beneficial consequences. (For example, see the paper [3].) Within the problem statement under study, we introduce the absolute maximum of the optimality criterion. Therefore, there can be no beneficial consequences, and the values $\rho(u)$ are interpreted as an undesirable risk due to using the control action $u$. By analogy with (3), we pose the risk minimization problem:

$$\rho(u^*) = \min_{u \in U} \rho(u). \quad (5)$$

Obviously,

$$u^* = \operatorname{Argmax}_{u \in U} \mathfrak{K}(u) = \operatorname{Argmin}_{u \in U} \rho(u). \quad (6)$$

Thus, in the case of complex systems control under uncertainty, **maximization of the optimality criterion (4)** is equivalent to **minimization of the risk (5): the identity (6) holds**.

## 2. THE STRUCTURE OF RISK MANAGEMENT PROBLEMS FOR COMPLEX SYSTEMS

Without loss of generality, the risk management problem for complex systems (see the statement in Section 1) can be decomposed into several particular, sequentially arising problems.

First, to determine the components of the system under study and its operating environment, it is necessary to solve *the identification problem*. Then, for the identified objects, it is necessary to determine a refer-

ence model of their behavior, i.e., solve *the modeling problem*. Next, it is necessary to compare the behavior of these objects with the reference model *to detect anomalies* in the latter. Finally, it is required *to predict* the situation considering the anomalies detected in the previous step.

A general solution of the risk minimization problem is based on solving the four problems mentioned. Note that these problems do not depend on the essence and structure of the system under study. At the same time, to build a technology for their solution, it is necessary to consider the subject area specifics, passing to applied and technological risk management problems. There are four classes of such problems; see Fig. 1.

Let us correlate the fundamental and applied risk management problems for complex systems (Fig. 1). When solving the fundamental problem of objects identification, we determine their most essential parameters. The set (e.g., Cartesian product) of the ranges of all objects parameters in the model is the state space of the system and its environment. Such a space is constructed by solving the applied parameterization problem.

Different objects of the system and its environment can exhibit different behavior. Therefore, to start solving the fundamental problem of behavior modeling, it is first necessary to solve the applied problem of classifying objects of the controlled system.

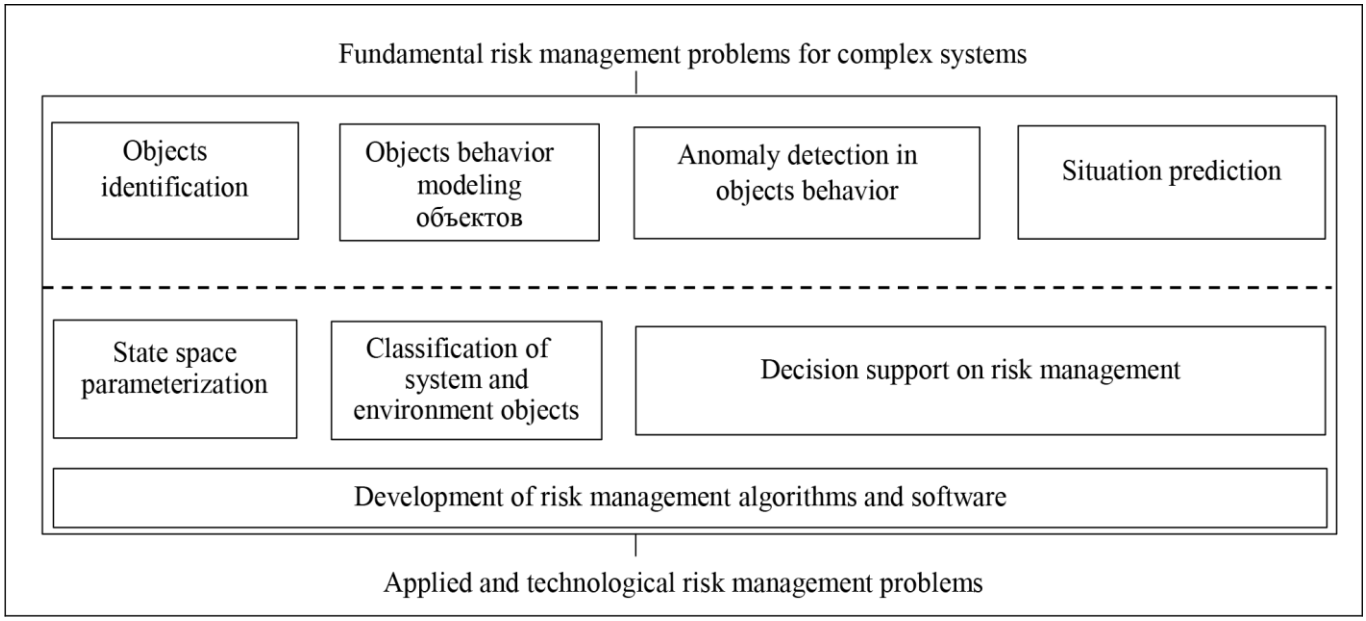



**Fig. 1. The hierarchical structure of risk management problems for complex systems.**

The fundamental problems of identifying behavioral anomalies and predicting the situation are associated at the application level with algorithms for classification, behavior modeling, anomaly detection, and prediction. In risk management, they are used together. At the same time, the development of decision-making algorithms is a separate applied problem that depends on the subject area but, as a rule, not on the implementation of other algorithms.

The natural computing algorithms and methods considered below are intended for solving the entire spectrum of the applied problems of classification, behavior modeling, anomaly detection, and prediction. From the authors' viewpoint, a good method should suit any of these problems, including the development of algorithms for application software. Further considerations will rest on this premise.

## 3. CLASSIFICATION OF NATURAL COMPUTING MODELS AND METHODS

When classifying natural computing, researchers often select as a basis the *biological process* (biological evolution, brain activity, sensory organs activity, etc.) underlying the algorithm or model of computations. This basis is convenient since it separates well the classes of algorithms. Its disadvantage lies in not answering the question: which problems can be solved by different classes of algorithms?

Another approach to classifying natural computing models and methods is selecting *the principle of algorithm development* as a basis. In the survey [4], all natural computing algorithms were divided into evolutionary, swarm, and ecological ones. Note that these

classes overlap: for example, artificial immune networks have characteristic features inherent in both swarm and evolutionary algorithms.

When considering natural computing models and algorithms with application to risk management in complex systems, it seems reasonable to select the classification bases corresponding to the specifics of problem (6). From the authors' viewpoint, the specifics are the variety of responses of such systems to external actions, not described by a finite set. Under uncertainty, the set of external actions $\Theta$ is not fully identified. Standard optimization techniques will only work in the identified part of this set. At the same time, a very important task is to minimize the risk in the event of new (previously unpredictable) external actions.

Considered within a single paradigm, the technological problem of developing algorithms and software requires a *universal* approach. In other words, the chosen approach should solve all applied risk management problems for complex systems. Therefore, the first (and most important) classification basis for natural computing models and methods will be universality. Since the matter concerns computational models, universality is understood primarily as the ability to implement an arbitrary algorithm (*universal computer*) or, at least, to calculate an arbitrary function with a given accuracy (*universal approximator*).

The risk management problem statement for complex systems assumes that the method or algorithm used must have a *formalism* adequate to the problem. In addition, since the systems under study often operate in a variable environment, the form of functions can (and will) change over time. This fact implies another requirement to the algorithm or method used, known as *learning capability*.

Thus, to solve the problem, it is advisable to adopt the following classification bases:

– universality (universal computer or universal approximator),

– the type of calculations (formal grammar, mathematical model, family of algorithms, element base, etc.),

– formalism (there is a formal model or not),

– learning capability (the ability to implement machine learning algorithms, the ability to implement adaptive algorithms, or no learning capability),

– technical implementation (hardware and software platforms).

## 4. NATURAL COMPUTING: A SURVEY

Natural computing models and methods include heuristic algorithms (or their families united by a common idea), some formal grammars, element bases of computers, and a series of mathematical models. The former include artificial immune systems, swarm intelligence, and amorphous and evolutionary computing. Formal grammars describe membrane computing [5] and Lindenmayer systems [6] (P systems and L-systems, respectively). The natural element bases of computers are founded on DNA molecules [7], Physarum amoebas [8], and some types of bacteria [9, 10]. Natural computing based on mathematical models includes cellular automata [11], artificial neural networks [12], and calculations with dynamic systems and chaos [13]. Let us briefly describe some of them.

### 4.1. Formal grammars

A formal **P system** is defined by an aggregate

$\Pi = (O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_o)$,

where:

$O$ is a non-empty finite alphabet of *objects*;

$C \subset O$ is the set of *catalysts*;

$\mu$ is a membrane structure composed of membranes $1, \ldots, m$ that determine the *regions* of the P system;

$w_1, \ldots, w_m$ are strings over the alphabet $O$ that describe *the multisets of objects* contained in regions $1, \ldots, m$, respectively;

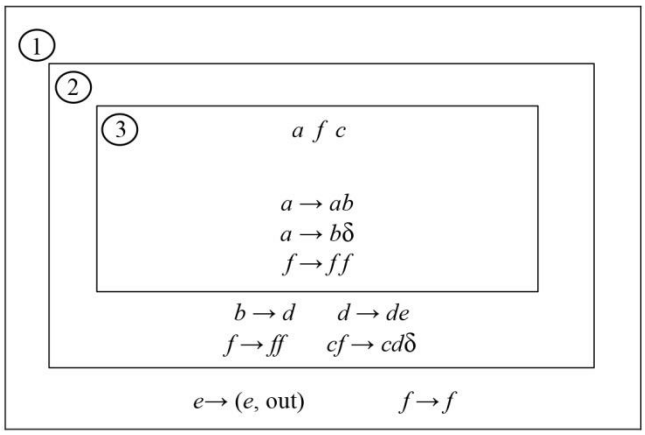
$R_1, \ldots, R_m$ is a finite *set of evolution rules* for regions $1, \ldots, m$, respectively;

finally, $i_o \in \{0, \cdots, m\}$ is the number of the region containing the result of calculations (zero result will be sent to the environment).

The evolution rules have the form $u \to v$ or $u \to v\delta$, where $u \in O^+$, $v \in (O \times Tar)^*$, and $Tar = \{here, in, out\}$. Here, $O^+$ denotes the set of all possible strings over the alphabet $O$, except the empty string $\lambda$, and $O^* = O^+ + \{\lambda\}$. Rules with an arbitrary action $u$ are called *cooperative*; rules with $u \in (O - C)$, *noncooperative*. Noncooperative rules with a catalyst, i.e., the ones of the form $cu \to cv$ or $cu \to cv\delta$, where $u \in (O - C)$, $c \in C$, and $v \in ((O - C) \times Tar)^*$, are called *catalytic*. The catalysts in such rules "help" other objects of the P system to evolve, not changing themselves and never moving between its regions.

The membrane structure and multisets of objects located in the regions bounded by the membranes define the configuration of the P system. The initial configuration is given by the membrane structure and the objects contained in the regions $(\mu, w_1, \ldots, w_m)$. In the course of the system evolution, applying the rules can change both the multisets of objects and the membrane structure. An example of the initial configuration of a P system is shown in Fig. 2.

Note that the rules of evolution in each step are applied in all regions of the system simultaneously.




**Fif. 2. Initial configuration of P system with three membranes, alphabet $O = \{a, b, c, d, e, f, \delta\}$, and the set of catalysts $C = \{c\}$.**

A detailed description of membrane computers can be found in [5].

A basic **L-system** consists of an alphabet $V$, an initial string (*axiom*) $\omega \in V^+$, where $V$ is the set of all nonzero-length strings over $V$, and the set of production rules $P$ of the form $p$: $a \to x$, $a \in V$, $x \in V^+$. For each symbol $a \in V$ not belonging to the left-hand side of $P$, the rule $a \to a$ holds. Such symbols are called *constants* or *terminal symbols*.

The rules from the set $P$ are applied iteratively, starting from the axiom (initial state). At each iteration, all symbols from the left-hand side of $P$ are replaced by the corresponding right-hand side. For example, the system $G = (V = \{a, b\}, \omega = a, P = \{a \to b, b \to ba\})$ produces the strings $a \to b \to ba \to ba\, b \to ba\, b\, ba \to ba\, b\, ba\, ba\, b \to \ldots$

Basic L-systems (called 0L-systems) involve no special symbols. More complex symbolic L-systems (SL-systems) recognize special substrings, i.e., reserved symbol operators. A detailed description of L-systems and particularly these operators, including several practical examples, can be found in [6].

### 4.2. Element bases

DNA computers encode information in nucleotide sequences. DNA molecules contain four nitrogenous bases: cytosine (C), guanine (G), adenine (A), and thymine (T). In the DNA computer model, nucleotide sequences are represented as strings over a finite alphabet $\{A, T, G, C\}$. The operations performed by the computer using various enzymes on DNA molecules are completely described by the rules of extended H-systems (also called splicing systems, see Chapter 7 of the book [14]).

Let

$$S = (V, \Sigma, A, R),$$

where: $V$ is a finite alphabet; $\Sigma \in V$ is a terminal alphabet; $A \subset V^*$ is the set of axioms; finally, $A \subset V^* \times V^* \times V^* \times V^*$ are splicing rules.

Each rule $r$ $r = (u_1, u_2, u_3, u_4) \in R$ is written as

$$r = u_1 \# u_2 \$ u_3 \# u_4;\ u_1, u_2, u_3, u_4 \in V^*;\ \#, \$ \notin V.$$

This expression is interpreted as follows: cut the string between consecutive entries of the substrings $u_1$ and $u_2$, $u_3$ and $u_4$; then glue the resulting fragments on the left of the first cut and on the right of the second one so that $u_1$ and $u_4$ turn out near each other. For example, consider some $x_1, x_2, y_1, y_2 \in V^*$ and strings $x = x_1 u_1 u_2 x_2$ and $y = y_1 u_3 u_4 y_2$. Applying the rule $r$ to the pair $(x, y)$ yields the string $z = x_1 u_1 u_4 y_2$.

The formal model of a DNA computer was described in detail in [15]. Bacterial computing [9, 10] is an implementation of such a computer in living organisms.

### 4.3. Mathematical models

**A cellular automaton** is defined by an aggregate $(\mathcal{L}, \mathcal{S}, \mathcal{N}, f)$, where:

$\mathcal{L} \subseteq \mathbb{Z}^D$ is the $D$-dimensional cellular space, i.e., a (possibly infinite) set of cells in $\mathcal{L}$ that forms a regular lattice;

$\mathcal{S}$ is the finite set of cells states;

$\mathcal{N} = (\vec{v_1}, \ldots, \vec{v_N})$ is the neighborhood vector from $N$ elements of the set $\mathcal{L}$ that connects a cell with its neighbors (for a cell located in $\vec{v} \in \mathcal{L}$, the neighbor cells are located in $(\vec{v} + \vec{v_i}) \in \mathcal{L}\ \forall i \in \{1, \ldots, N\}$);

Finally, $f: S^N \to S$ is a local transition rule determining the cell state at the next time instant, $f(a_1, \ldots, a_N)$, where $a_1, \ldots, a_N$ are the states of its $N$ neighbors.

A *configuration* of a cellular automaton is a global state $C: \mathcal{L} \to \mathcal{S}$. Let $\mathcal{S}^{\mathcal{L}}$ be the set of all configurations. Then a mapping $G: \mathcal{S}^{\mathcal{L}} \to \mathcal{S}^{\mathcal{L}}$ is called a global transition function, and a cellular automaton can be defined by the aggregate $\left(\mathcal{L},\ \mathcal{S},\ \mathcal{S}^{\mathcal{L}}, G\right)$.

Computing based on a cellular automaton is performed by sequentially calculating the transition function. As a rule, a cellular automaton is assumed to be synchronous (all cells change their state simultaneously) and homogeneous (all cells use a single transition rule). In this case, if a configuration $y = (y_{\vec{v}})_{\vec{v} \in v\mathcal{L}}$ follows a configuration $x = (x_{\vec{v}})_{\vec{v} \in v\mathcal{L}}$ (that is, $y = G(x)$), $y$ will be the result of calculating the following expression for each $\vec{v} \in \mathcal{L}$:

$$y_{\vec{v}} = G(x)_{\vec{v}} = G(x_{\vec{v}}) = f\left(x_{\vec{v}+\vec{v_1}}, \ldots, x_{\vec{v}+\vec{v_N}}\right).$$

The model, types, and properties of cellular automata were considered in detail in the book [16].

The formal model of **an artificial neuron** is written as

$$s = \mathbf{w} \cdot \mathbf{x} + b,\ z = g(s),$$

where: $\mathbf{x} \in \mathbb{Z}_{\{0,1\}}^{N}$ denotes the vector of *input data*; $\mathbf{w}$ denotes the vector of *weights*; $b$ is the displacement; $g(\cdot)$ is *an activation function*; finally, $z$ denotes *the output*. When forming an artificial neural network, individual neurons $s$ are combined into a layer

$$\mathbf{s} = \mathbf{W}\mathbf{x} + \mathbf{b},\ \mathbf{z} = g(\mathbf{s}),$$

where $\mathbf{W}$ denotes the matrix of weights. A neural network can have several interconnected layers.

An artificial neural network is trained by adjusting the weights, as a rule, via minimizing some loss functional: $L(\mathbf{w}) \to \min_{\mathbf{w}}$.

The evolution of artificial neural network architectures, learning modes, and applications was discussed in the survey [17].

The computing model with dynamic systems and chaos is based on the concept of a chaotic element: *a chaotic chip* or *a chaotic processor* [18]. Let the state of such an element be determined by the variable $x$. The logical gate implemented by this element is constructed in the following way.

- The input signals are:

$x \to x_0 + X_1 + X_2$ for the binary logical operations (NAND, NOR, XOR, AND, OR, and XNOR);

$x \to x_0 + X$ for the unary operations (NOT),

where $x_0$ denotes the initial system state and

$$X = \begin{cases} 0 & \text{for } I = 0, \\ V_{in} > 0 & \text{for } I = 1, \end{cases}$$

where $V_{in}$ is a positive constant.

- Dynamic updating is used, i.e., $x \to f(x)$, where $f(x)$ is a linear function.
- A threshold mechanism is used for obtaining the output signal $Z$:

$$Z = \begin{cases} 0 & \text{if } f(x) \le E, \\ f(x) - E & \text{if } f(x) > E, \end{cases}$$

where $E$ is a threshold.

The output value is interpreted as logical zero if $Z = 0$ and as logical one if $Z > 0$.

A feature of chaotic elements is the ability to change the type of the logical gate they implement using control of the values $(x_0, E)$. A classical example is a chaotic element with $f(x) = 4x(1 - x)$ and $V_{in}$=1/4. This element implements one of the logical operations AND, OR, XOR, NAND, or NOT, depending on the values $(x_0, E)$; see Table 1.

The chaotic computing paradigm was considered in detail in the paper [18].

*Table 1*

**Values $x_0$ and $E$ corresponding to different logical operations of chaotic element with $f(x) = 4x(1 - x)$ and $V_{in}$=1/4**

| Logical operation | AND | OR | XOR | NAND | NOT |
|---|---|---|---|---|---|
| $x_0$ | 0 | 1/8 | ¼ | 3/8 | 1/2 |
| $E$ | 3/4 | 11/16 | ¾ | 11/16 | 3/4 |

The other approaches are either unique in natural computing or combine very heterogeneous elements and therefore cannot be identified with any of the groups mentioned.

### 4.4. Universality of natural computing

Consider the universality of different types of natural computing.

Cellular automata and artificial neural networks are universal computers. Smith's classical work [19] was devoted to the Turing-complete cellular automaton. The Turing completeness theorem for fully connected recurrent artificial neural networks with sigmoidal activation functions was proved in the paper [20].

The computing paradigm with nonlinear dynamic systems (chaos) emerged in response to the limitations of transistors as a traditional element base. Logical elements (gates) based on transistors cannot be changed after the hardware implementation. Programming involves switching between several different single-purpose elements. A chaotic element is capable of transforming into various logical elements using a threshold-based morphing mechanism. The work [21] described the implementation of the AND, OR, NOT, and XOR gates, substantiating the universality of chaotic chips.

Collision-based computing [22], also called ballistic computing, free space computing, and billiard computing in the literature, implement logical circuits using a homogeneous unstructured environment with mobile localizations. These can be gliders in cellular automata, solitons in optical systems, or wave fragments in excitable chemical systems. Logical truth corresponds to the presence of localization; logical false, to the absence of localization. When two or more moving localizations collide, they change their velocity and (or) state vectors. The post-collision paths and (or) states of localizations result from logical operations implemented by a collision. The equivalence of the billiard computing model based on a two-dimensional block cellular automaton to a reversible Turing machine was established in [23].

The term "reaction-diffusion computing" can be interpreted as a computational model based on the corresponding family of differential equations and as a chemical computer in which data are represented by concentrations of different chemical elements and processing is performed using chemical reactions (e.g., the Belousov–Zhabotinsky reaction); see [24, 25]. On the other hand, reaction-diffusion equations are often used to model other computational environments, particularly chaos and collision-based computing. The paper [26] demonstrated the fundamental possibility of universal calculations in chemical computers under limited resources. Note that a similar result was obtained therein for Physarum computing, discussed below.

Now let us consider formal grammars. Membrane computing (also known as P systems [5]) involves the membrane concept based on the simplest analogy with biological cells. During the operation of such systems, the objects of calculations (the sets of symbolic multisets) and, moreover, the membrane structures themselves evolve. This peculiarity is inherent in P systems. Each membrane structure (cell) is treated as a separate computational element, and the principle of maximum parallelism is employed. Consequently, such a grammar allows writing P-systems that solve NP-complete problems in polynomial time (due to the exponential growth of the number of membranes and, accordingly, parallelism). Computational completeness was proved for some P systems. In particular, this was done for communicative membrane systems [27]. (In such systems, actions over an object are performed when it passes through the membrane.) In the paper [28], a similar assertion was derived for several classes of P systems with catalysts (symbols in a multiset that cause the application of some rule, not "consumed" by it). In 2009, Turing completeness was shown for an extension of P systems, the so-called mutual mobile membrane systems, in the case of three membranes [29].

Lindenmayer systems (L-systems) were originally proposed by botanist A. Lindenmeier [30] as a formal language for describing algal growth. Later on, the approach was developed into a formal grammar. In 1991, L-systems were first used for expanding some elementary functions into a Taylor series [6]. A corresponding compiler and a visualization subsystem for the IBM PC platform were created soon [31]. However, the generation of fractal structures remains the main application of Lindenmayer systems. The computational completeness of such systems is not discussed in the literature.

A separate trend in the development of natural computing is searching for new (alternative) computational element bases. In 1994, it was proposed to use a DNA molecule [7]. An advantage of the DNA approach is the simultaneous generation of all possible solutions of combinatorial problems (e.g., Hamiltonian path problems in directed graphs) using known biochemical reactions. Then the molecule string encoding the desired response can be quickly filtered out. However, when scaling the method proposed by L. Adleman, the number of DNA molecules needed to find a solution grows exponentially with the problem dimensionality. This fact imposes physical restrictions on the computational power of such a computer. Nevertheless, already in 1999, it was established that DNA computers can be universal computers [32]. In 2017, a design of a DNA computer implementing a nondeterministic universal Turing machine was proposed [33].

In 2010, using Adleman's ideas, a group of researchers created a bacterial computer based on genetically modified E. coli bacteria [10]. In contrast to the earlier experiments [9], the researchers placed the DNA sequences encoding the problem to the breaks in the DNA strands of genes encoding fluorescent proteins. Note that the inserted fragments were framed by the so-called hix sites. As a result, the effect of site-specific inversion of bacterial DNA in the presence of a special protein, DNA-invertase Hin, was used. During the inversion of the nucleotide strand, the encoding fluorescent proteins are restored, and the bacteria that "found" the correct solution glow under the microscope.

Another approach to the creation of non-traditional computers is to use some special properties of living organisms. The most famous experiment involves the slime mould (amoeba) Physarum polycephalum, which tends to take a shape minimizing the sunlight effect on it. The paper [34] described the application of such an "amoebic computer" to the approximate solution of the NP-complete traveling salesman problem. As demonstrated therein, the amoeba solves this problem in linear time. However, to calculate the lighting scheme of the amoeba, a recurrent neural network is used, whose dynamics are determined by a weight matrix with $n^4$ elements ($n$ denotes the number of cities). Therefore, the gain in time is not obvious for large dimensions. The universality of such a computer has not yet been investigated.

Finally, we consider the families of heuristic and metaheuristic algorithms commonly related to natural computing. Artificial immune systems include several classes of such algorithms, created by analogy with their natural implementation in the immune systems of vertebrates. Despite no generally accepted formalization (the one proposed in [35] has not become wide-

spread), the results of [36, 37] suggest that a universal approximator can be based on an artificial immune system.

Swarm intelligence algorithms usually include system models in which many agents interact locally with the environment and with each other. The agents obey some fairly simple rules of behavior. However, the entire multi-agent system exhibits complex ("intelligent") behavior. In practice, such algorithms are used for solving various optimization problems. For example, see the survey [38]. The universality of such algorithms (e.g., as a universal approximator) has apparently not been raised in the literature.

The term "amorphous computing" was introduced by a group of MIT researchers in 1996. It refers to a class of computing devices consisting of numerous cheap, almost identical information processing units. In this case, cheapness is an essential property: among ther things, such devices were intended for the wholesale production of smart structural materials as an additive ("improver"). The possibility of universal calculations on an amorphous computer composed of asynchronously operating finite probabilistic automata was first shown in [39]. Several amorphous computing systems from the class of universal computers were considered in [40].

Along with the fuzzy logic and swarm intelligence models considered previously, evolutionary computing belongs to a large class of the so-called "soft computing" with approximate models. Technically, evolutionary computing is a family of global optimization algorithms based on the idea of biological evolution. The family of candidate solutions forms a "population" that is gradually improved by selection or random "mutations." The process stops when the solutions reach the required level of accuracy.

The forms of natural computing, the types of corresponding calculations, and their universality are described in Table 2, including the relevant references.

## 5. NATURAL COMPUTING FOR RISK MANAGEMENT IN COMPLEX SYSTEMS: ANALYSIS OF POSSIBLE USE

As noted earlier, natural computing tools suitable for risk management in complex systems must, first of all, be universal, have a formal mathematical model, and, finally, demonstrate a high level of adaptiveness. In terms of applications, it is also important that the model or algorithm have software implementation (to be used when developing specialized software) or hardware implementation (to speed up operation). The latter requirement becomes crucial for risk management in information security (e.g., when creating traffic analyzers). In the previous section, we have selected the types of natural computing with proven universality. Let us check whether they match the other criteria.

*Table 2*

**Well-known natural computing methods and algorithms and their universality**

| Form of natural computing | Type of calculations | Universality |
|---|---|---|
| Cellular automata | Mathematical model, element base | Universal computer [19] |
| Artificial neural networks | Mathematical model, element base | Universal computer [20] |
| Computing with dynamic systems and chaos | Mathematical model, element base | Universal computer [21] |
| Collision-based computing | Computational model | Universal computer [23] |
| Reaction-diffusion computing | Mathematical model, element base | Universal computer [26] |
| Membrane computing (P systems) | Formal grammar | Universal computer [27–29] |
| Lindenmayer systems (L-systems) | Formal grammar | Applicable to symbolic calculations [6] |
| DNA computing | Element base | Universal computer [32, 33] |
| Bacterial computing | Element base | Universal computer (possibly) |
| Physarum computing | Element base | Universal computer [26] |
| Artificial immune systems | Family of algorithms | Universal approximator (possibly) [36, 37] |
| Swarm intelligence | Family of algorithms | Universal approximator (possibly) |
| Amorphous computing | Family of algorithms | Universal computer [39, 40] |
| Evolutionary computing | Family of metaheuristic algorithms | Not applicable |

According to the Curtis–Hedlund–Lyndon theorem for cellular automata, proved in 1969, transitions between any two shift spaces can be determined by a uniformly local rule [41]. The class of adaptive stochastic cellular automata allows implementing adaptive algorithms [42]. Technically, the formal automaton model is implemented on a conventional computer: free software modules are available in Python and Wolfram (Mathematica). Thus, adaptive control systems can be designed based on this type of computing.

Artificial neural networks received a formal model within finite automata theory back in 1956; see [43]. One year later, the classical work was published describing the learning algorithm for an artificial neural network based on a two-layer perceptron in which one layer is hidden and untrainable [44]. Many programmed neural network architectures are currently available for solving various classes of problems (mainly recognition and classification). Their hardware implementations are constantly being improved; for example, see the survey [45].

Chaotic calculations involve models of various nonlinear dynamic systems [13], which give the necessary formalism. The behavior of such systems can be numerically simulated, e.g., in MATLAB. Work is now underway towards the design of a chaotic processor. In particular, the paper [18] described the operation of chaotic transistors. The same research team patented the architecture of an arithmetic-logic device based on such transistors [46]. However, this class of systems does not exhibit learning capability.

Collision-based computing systems are often called billiard computers, and their model is often called the formal billiard model. The paper [47] described all currently used mobile and stationary localizations ("balls" and "tables," respectively). Such systems have no learning capability. The models of such computers can be constructed in MATLAB. There is no hardware implementation yet; the most recent known publication in this area [48] presented the idea of a chemical transistor using chemical wave fragments as stationary localizations.

The Kolmogorov – Petrovsky – Piskunov equation is considered the first formal reaction-diffusion system model in the one-dimensional case [49]. More complex models are required to describe environments suitable for computing. One example is a chemical computer based on the Belousov – Zhabotinsky reaction [26, 50]. Such systems are not trainable. ReaDDy 2 library [51] is the contemporary software implementation of the "reaction-diffusion" computing model. Also, some experiments were carried out with the programming of a chemical computer [24].

The main component of a membrane computer (membrane structure) was formally described by Paun, the pioneer of P systems [5]. Such a grammar allows writing adaptive algorithms. However, the membrane computer has no learning capability. Its software implementation is the P-Lingua programming language [52], a plug-in to Eclipse IDE. There are no hardware implementations.

Natural computing systems with alternative bases (DNA, bacterial, and Physarum computers) have no unique computational model. They use classical ones instead: a Turing machine, nondeterministic finite automata, or others, depending on the researcher's preferences. Hence, such computers are without "innate" learning capability, although they allow implementing adaptive and even self-learning algorithms. All such computers have hardware implementation; see the references in Table 3.

Most researchers still consider artificial immune systems as a set of heuristic algorithms. In 1998, an attempt was undertaken to base such systems on the formal peptide model [35], but this formalism is not yet generally accepted. Perhaps this circumstance hinders the development of software modules for implementing artificial immune systems-based solutions: several software products found by the authors (see Table 3) are not being developed at the moment. Immune networks have the "innate" learning capability and are comparable by the potential to artificial neural networks; for example, see [53].

Swarm intelligence algorithms do not rest on a single formal model of a swarm's element (particle, agent, etc.). They are united by the principle of local interaction of elements with each other and the environment. Accordingly, such algorithms cannot be trained, although some prediction problems are solved using swarm algorithms. The most popular representative of this family – the particle swarm method – has several software implementations in the form of plug-ins for MATLAB [54] and SCILAB [55]. The other swarm algorithms are less common and usually implemented by researchers to solve particular narrow problems.

Amorphous computing also does not imply an underlying formal model: it denotes a class of computing systems with a very high degree of parallelism. Specialized programming languages were developed to facilitate the programming of such systems (e.g., sensor networks). For details, see Table 3.

*Table 3*

**Natural computing methods and algorithms and their characteristics: the presence of formal model, its learning capability, and examples of technical implementation**

| Form of natural computing | Formalism | Learning capability | Technical implementation |
|---|---|---|---|
| Cellular automata | Yes [41] | Adaptiveness [42] | Software platform (Python, Wolfram Mathematica) |
| Artificial neural networks | Yes [43] | Yes [44] | Software implementations (PyTorch, TensorFlow, etc.) and hardware implementations (see the survey [45]) |
| Computing with dynamic systems and chaos | Yes [13] | No | Software implementation (MATLAB) and hardware implementation [18] |
| Collision-based computing | Yes [47] | No | Software implementation (MATLAB) |
| Reaction-diffusion computing | Yes [49, 50] | No | Software implementation (ReaDDy library for Python and Java [51]) |
| Membrane computing (P systems) | Yes [5] | No | Software implementation (P-lingua language [52]) |
| DNA computing | Yes [15] | No | DNA self-assembly [56] |
| Bacterial computing | No | No | Genetically modified E. Coli bacteria [9] |
| Physarum computing | No | No | Amoeba-based computing system [34] |
| Artificial immune systems | Yes [35] | Yes [53] | Software implementation (Jisys [57], iNet Framework [58] and libtissue [59] libraries) |
| Swarm intelligence | No | No | Software implementation for PSO [54, 55] |
| Amorphous computing | No | No | GPL (Growing Point Language) [60] and Proto language [61] |

## 6. RISK MANAGEMENT WITH NATURAL COMPUTING

The formal analysis presented above identifies natural computing methods suitable for solving risk management problems in complex systems. These are cellular automata, artificial neural networks, and artificial immune systems.

To compare the prevalence of these approaches in fundamental and applied risk management problems for complex systems, we will consider the number of Google Scholar publications on the subject (search results for the requests containing the problem statement with synonymous constructs and the method name; see Tables 4 and 6). Note that the effectiveness of the methods in particular problems will be neither assessed nor compared. The number of publications rather shows the "popularity" of the approach in the scientific community, indirectly characterizing the depth of development.

The requests are structured as follows. The text of an appropriate publication should contain the problem statement (e.g., "identification" for the identification problem; "forecast OR prediction" for the prediction problem). Also, the text of an appropriate publication should contain the corresponding natural computing method or model (e.g., "artificial immune system" for artificial immune systems): its full name or generally accepted abbreviation. Synonyms in the requests are separated by the OR operator.

Note that the high results obtained for artificial neural networks could be even higher: Google Scholar limits the maximum execution time of each search request and stops viewing the publication index after it. Thus, this approach to risk management in complex systems is used much more often than the other artificial intelligence models. The conclusion concerns both fundamental and applied and technological problems (see Table 7).

Cellular automata are inferior in the frequency of use to both neural and immune networks. Rare publications involve them in identifying anomalies or as a component of an information system. Nevertheless, anomaly detection is successfully solved by cellular automata. We mention the paper [67] among the modern research on this topic.

Artificial immune networks have been developing since recently, showing themselves to be a rather promising approach. This explains their position between cellular automata and neural networks by the

*Table 4*

**Google Scholar publications on fundamental risk management problems for complex systems**

| Class of problems | Search requests and examples of search results |
|---|---|
| Cellular automata | |
| Identification | intext: “identification” allintitle: “by cellular automata” OR “cellular automata for” – “identification of cellular automata” – “identification of optimal cellular automata” – “identification number” [62, 63] |
| Behavior modeling | allintext: “behavior” “activity” “modeling” OR “simulation” allintitle: “by cellular automata” OR “cellular automata for” [64, 65] |
| Anomaly detection | allintext: “anomaly detection” allintitle: “by cellular automata” OR “cellular automata for” [66, 67] |
| Prediction | allintext: “forecast” OR “prediction” allintitle: “by cellular automata” OR “cellular automata for” [68, 69] |
| Artificial neural networks | |
| Identification | identification “artificial neural network” OR “neural network” OR “deep learning” – “identification of ANN” – “identification of neural network” – “identification number” – “neural network identification” [70, 71] |
| Behavior modeling | behavior activity (simulation OR modelling) AND (“artificial neural network” OR “neural network” OR “deep learning”) – “neural network dynamics” – “neural network training” [72, 73] |
| Anomaly detection | “anomaly detection” AND (“artificial neural network” OR “neural network” OR “deep learning”) [74, 75] |
| Prediction | (forecast OR prediction) AND (“artificial neural network” OR “neural network” OR “deep learning”) [76, 77] |
| Artificial immune networks | |
| Identification | allintext: “identification” “computing” “artificial immune system” allintitle: “AIS” OR “immune system” OR “artificial immune” – “identification of AIS” – “identification of artificial immune system” – “identification number” – “immune system identification” [78, 79] |
| Behavior modeling | allintext: “behavior” “activity” “modeling” OR “simulation” “artificial immune system” allintitle: “AIS” OR “immune system” [80, 81] |
| Anomaly detection | allintext: “anomaly detection” “artificial immune system” allintitle: “AIS” OR “immune system” [82, 83] |
| Prediction | allintext: “forecast” OR “prediction” “artificial immune system” allintitle: “AIS” OR “immune system” [84, 85] |

number of publications on the same classes of problems. However, their use in information systems is constrained by the absence of a generally accepted formalism and, accordingly, software implementation in the form of a software library.

Again, the search engine results for the requests (Tables 5 and 7) do not substantiate the comparative “suitability” of the model for a particular class of problems but rather describe the distribution of the scientific community’s preferences. A relatively small number of references indicates that a particular method or model is less popular. The reasons may be insufficient studies of the model and the lack of convenient software tools for its use.

*Table 5*

**Using artificial intelligence models and methods in fundamental risk management problems for complex systems (by the number of Google Scholar publications)**

| Problem | Cellular automata | Artificial neural networks | Artificial immune networks |
|---|---|---|---|
| Identification | 2 390 | 17 800 | 5 720 |
| Behavior modeling | 1 630 | 20 700 | 2 550 |
| Anomaly detection | 56 | 17 200 | 4 170 |
| Prediction | 4 220 | 18 000 | 7 520 |

*Table 6*

**Google Scholar publications on applied and technological risk management problems for complex systems**

| Class of problems | Search requests |
|---|---|
| Cellular automata | |
| Classification | allintext: “classification of” allintitle: “by cellular automata” OR “cellular automata for” [86, 87] |
| Decision support | allintext: “decision support” allintitle: “by cellular automata” OR “cellular automata for” [88, 89] |
| Development of information and control systems | allintext: “information system development” OR “software development” “control” allintitle: [96, 97] |
| Artificial neural networks | |
| Classification | “classification of” AND (“artificial neural network” OR “neural network” OR “deep learning”) – “ANN classification” – “neural network classification” – “classification of neural” [92, 93] |
| Decision support | “decision support” AND (“artificial neural network” OR “neural network” OR “deep learning”) [94, 95] |
| Development of information and control systems | “control” AND (“software development” OR “system development”) AND (“artificial neural network” OR “neural network” OR “deep learning”) [96, 97] |
| Artificial immune networks | |
| Classification | allintext: “classification of” “artificial immune system” allintitle: “AIS” OR “immune system” – “AIS classification” – “classification of artificial immune” – “classification of immune” – “immune system classification” [98, 99] |
| Decision support | allintext: “decision support” “artificial immune system” allintitle: “AIS” OR “immune system” [100, 101] |
| Development of information and control systems | allintext: “information system development” OR “software development” “control” “artificial immune system” allintitle: “AIS” OR “immune system” [102, 103] |

*Table 7*

**Using artificial intelligence models and methods in applied and technological risk management problems for complex systems (by the number of Google Scholar publications)**

| Problem | Cellular automata | Artificial neural networks | Artificial immune networks |
|---|---|---|---|
| Classification | 4 330 | 18 200 | 4 780 |
| Decision support | 1 050 | 17 800 | 2 210 |
| Development of information and control systems | 242 | 17 300 | 628 |

## CONCLUSIONS

The use of analytical methods for risk management in complex systems shows predictable results. However, control performance directly depends on the structure of the controlled system. For multi-agent heterogeneous open systems, an analytical solution of risk management problems can take a long time. Moreover, the resulting solution is often effective only in a small domain of the state space.

Natural computing solutions are interesting due to high adaptiveness, a consequence of the complexity of natural systems. Nevertheless, not all of these approaches have the required level of abstractness, adaptiveness, and learning capability.

This paper has analyzed the known natural computing models and methods with application to both fun-

damental and applied and technological risk management problems in complex systems. Three models have been identified that fully meet all the criteria: artificial neural networks, cellular automata, and artificial immune networks.

Artificial neural networks have been used for risk management in complex systems for many years and have proven their effectiveness. The other two approaches are less common, but their potential cannot be considered lower. The most probable reasons hindering their development are no generally accepted formalism for artificial immune networks and no learning capability for cellular automata.

It seems promising to develop risk management methods for complex systems based on cellular automata and artificial immune systems.

## REFERENCES


1. *Handbook of Natural Computing*, Rozenberg, G., Bäck, T., and Kok, J.N., Eds., Berlin: Springer Berlin Heidelberg, 2012.
2. Kalashnikov, A.O., Managing the Information Risks of Organizational Systems: A General Problem Statement, *Informatsiya i Bezopasnost'*, 2016, vol. 19, no. 1, pp. 36–45. (In Russian.)
3. Kononov, D.A., Studying the Security of Control Systems by Analyzing Their System Parameters, *Materialy 28-oi Mezhdunarodnoi konferentsii "Problemy upravleniya bezopasnost'yu slozhnykh sistem"* (Proceedings of 28th International Conference on Complex Systems Security Control), December 16, 2020, Moscow, Kalashnikov, A.O. and Kul'ba, V.V., Eds., Moscow: Trapeznikov Institute of Control Sciences, 2020, pp. 102–108. (In Russian.)
4. Nemade, M.N. and Rane, M.D., A Review on Bio-Inspired Computing Algorithms and Application, *Proceedings of National Conference on Recent Trends in Computer Science and Information Technology* (*NCRTCSIT-2016*), Nagpur, 2016, pp. 12–19.
5. Paun, G., Introduction to Membrane Computing, in *Applications of Membrane Computing*, Ciobanu, G., Paun, G., and Perez-Jimenez, M.J., Eds., Berlin–Heidelberg: Springer, 2006, pp. 1–42.
6. Goel, N.S. and Goodwin, M.D., Symbolic Computation Using L-systems, *Applied Mathematics and Computation*, 1991, vol. 42, no. 3, pp. 223–253.
7. Adleman, L.M., Molecular Computation of Solutions to Combinatorial Problems, *Science*, 1994, vol. 266, no. 5187, pp. 1021–1024.
8. Adamatzky, A., *Physarum Machines: Computers from Slime Mould*, vol. 74, Singapore: World Scientific, 2010.
9. Haynes, K., Broderick, M., Brown, A., et al., Engineering Bacteria to Solve the Burnt Pancake Problem, *Journal of Biological Engineering*, 2008, vol. 2, article no. 8.
10. Poet, J.L., Campbell, A.M., Eckdahl, T.T., and Heyer, L.J., Bacterial Computing, *XRDS: Crossroads, The ACM Magazine for Students*, 2010, vol. 17, no. 1, pp. 10–15.
11. Codd, E.F., *Cellular Automata*, New-York: Academic Press, 2014.
12. Goodfellow, I., Bengio, Y., and Courville, A., *Deep Learning*, Cambridge: The MIT Press, 2016.
13. Ditto, W.L., Miliotis, A., Murali, K., and Sinha, S., The Chaos Computing Paradigm, in *Reviews of Nonlinear Dynamics and Complexity*, Schuster, H.G., Ed., Weinheim, Germany: Wiley-VCH, 2010, pp. 1–35.
14. Paun, G., Rozenberg, G., and Salomaa, A., *DNA Computing. New Computing Paradigma*, Berlin–Heidelberg: Springer, 1998.
15. Krasinski, T. and Sakowski, S., A Theoretical Model of the Shapiro Finite State Automaton Built on DNA, *Theoretical and Applied Informatics*, 2006, vol. 18, pp. 161–174.
16. Hadeler, K.-P. and Müller, J., *Cellular Automata: Analysis and Applications*, Cham: Springer, 2017.
17. Makarenko, A.V., Deep Neural Networks: Origins, Development, Current Status, *Control Sciences*, 2020, no. 2, pp. 3–19. (In Russian.)
18. Ditto, W.L., Miliotis, A., Murali, K., et al., Chaogates: Morphing Logic Gates That Exploit Dynamical Patterns, *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 2010, vol. 20, no. 3, article no. 037107.
19. Smith, A., Simple Computation-Universal Cellular Spaces, *Journal of the ACM*, 1971, vol. 18, no. 3, pp. 339–353.
20. Siegelmann, H. and Sontag, E., Turing Computability with Neural Nets, *Appl. Math. Lett.*, 1991, vol. 4, no. 6, pp. 77–80.
21. Munakata, T., Sinha, S., and Ditto, W.L., Chaos Computing: Implementation of Fundamental Logical Gates by Chaotic Elements, *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 2002, vol. 49, no. 11, pp. 1629–1633.
22. Adamatzky, A., *Collision-Based Computing*, London: Springer-Verlag, 2002.
23. Durand-Lose, J., About the Universality of the Billiard Ball Model, *Multiple-Valued Logic*, 1998, vol. 6, no. 5, pp. 118–132.
24. Adamatzky, A., Programming Reaction-Diffusion Processors, in *Unconventional Programming Paradigms*, Berlin–Heidelberg: Springer, 2005, pp. 33–46.
25. Gorecki, J., Gizynski, K., Guzowski, J., et al., Chemical Computing with Reaction-Diffusion Processes, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 2015, vol. 373, no. 2046, pp. 20140219.
26. Adamatzky, A., de Lacy Costello, B., and Shirakawa, T., Universal Computation with Limited Resources: Belousov–Zhabotinsky and Physarum Computers, *International Journal of Bifurcation and Chaos*, 2008, vol. 18, no. 8, pp. 2373–2389.
27. Alhazov, A., Margenstern, M., Rogozhin, V., et al., Communicative P Systems with Minimal Cooperation, *The International Workshop on Membrane Computing* (*WMC5*), Milan, 2004. Revised papers published in *LNCS*, vol. 3365, Berlin: Springer, 2005, pp. 162–178.
28. Freund, R., Kari, L., Oswald, M., and Sosik, P., Computationally Universal P Systems without Priorities: Two Catalysts Are Sufficient, *Theoretical Computer Science*, 2005, vol. 330, no. 2, pp. 251–266.
29. Aman, B. and Ciobanu, G., Turing Completeness Using Three Mobile Membranes, in *Unconventional Computation* (*UC 2009*), Calude, C.S., Costa, J.F., Dershowitz, N., Freire, E., and Rozenberg, G., Eds., *Lecture Notes in Computer Science*, vol. 5715, Berlin–Heidelberg: Springer-Verlag, 2009, pp. 42–55.
30. Lindenmayer, A., Mathematical Models for Cellular Interaction in Development I-II, *J. Theoret. Biology*, 1968, no. 18, pp. 280–315.
31. Goel, N. and Rozehnal, I., A High-Level Language for L-systems and Its Applications, in *Lindenmayer Systems: Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology*, Rozenberg, G. and Salomaa, A., Eds., Berlin: Springer-Verlag, 1992, pp. 231–251.

32. Winfree, E., Yang, X., and Seeman, N.C. Universal Computation via Self-assembly of DNA: Some Theory and Experiments, *Proceedings of the 2nd DIMACS Workshop on DNA Based Computers*, June 10–12, 1996, Princeton, 1996, pp. 191–213.
33. Currin, A., Korovin, K., Ababi, M., et al., Computing Exponentially Faster: Implementing a Non-Deterministic Universal Turing Machine Using DNA, *Journal of the Royal Society. Interface*, 2017, vol. 14, no. 128, article no. 20160990.
34. Zhu, L., Kim, S.-J., Hara, M., and Aono, M., Remarkable Problem-Solving Ability of Unicellular Amoeboid Organism and Its Mechanism, *Royal Society Open Science*, 2018, vol. 5, no. 12, article no. 180396.
35. Tarakanov, A. and Dasgupta, D., A Formal Model of an Artificial Immune System, *Biosystems*, 2000, vol. 55, no. 1-3, pp. 151–158.
36. Gong, M., Jiao, L., and Zhang, X., A Population-Based Artificial Immune System for Numerical Optimization, *Neurocomputing*, 2008, vol. 72, no. 1–3, pp. 149–161.
37. Ülker, E. and Arslan, A., Automatic Knot Adjustment Using an Artificial Immune System for B-Spline Curve Approximation, *Information Sciences*, 2009, vol. 179, no. 10, pp. 1483–1494.
38. Chakraborty, A. and Kar, A.K., Swarm Intelligence: A Review of Algorithms, in *Nature-Inspired Computing and Optimization. Modeling and Optimization in Science and Technologies*, Patnaik, S., Yang, X.S., and Nakamatsu, K., Eds., vol. 10, Cham: Springer, 2017, pp. 475–494.
39. Wiedermann, J., Computability and Non-computability Issues in Amorphous Computing, in *Theoretical Computer Science* (*TCS 2012*), *Lecture Notes in Computer Science*, Baeten, J.C.M., Ball, T., and de Boer, F.S., Eds., vol. 7604, Berlin– Heidelberg: Springer, 2012, pp. 1–9.
40. Wiedermann, J. and Petrů, L., On the Universal Computing Power of Amorphous Computing Systems, *Theory of Computing Systems*, 2009, vol. 45, no. 4, pp. 995–1010.
41. Hedlund, G.A., Endomorphisms and Automorphisms of the Shift Dynamical System, *Mathematical Systems Theory*, 1969, vol. 3, no. 4, pp. 320–375.
42. Lee, Y.C., Qian, S., Jones, R.D., et al., Adaptive Stochastic Cellular Automata: Theory, *Physica D: Nonlinear Phenomena*, 1990, vol. 45, no. 1–3, pp. 159–180.
43. Kleene, S.C., Representation of Events in Nerve Nets and Finite Automata, in *Automata Studies* (AM-34), vol. 34, Princeton: Princeton University Press, 1956, pp. 3–42.
44. Rosenblatt, F., The Perceptron – a Perceiving and Recognizing Automaton, *Report 85–460–1*, Cornell Aeronautical Laboratory, 1957.
45. Schuman, C.D, Potok, T.E., Patton, R.M., et al., A Survey of Neuromorphic Computing and Neural Networks in Hardware, *arXiv:1705.06963*, 2017.
46. Ditto, W.L., Sinha, S., and Murali, K., Method and Apparatus for a Chaotic Computing Module Using Threshold Reference Signal Implementations, US Patent WO/2005/036353, 2006.
47. Adamatzky, A. and Durand-Lose, J., Collision-Based Computing, in *Handbook of Natural Computing*, Rozenberg, G., et al., Eds., Berlin–Heidelberg: Springer-Verlag, 2012, pp. 1950–1978.
48. Toth, R., Stone, C., de Lacy Costello, B., et al., Simple Collision-Based Chemical Logic Gates with Adaptive Computing, *International Journal of Nanotechnology and Molecular Computation*, 2009, vol. 1, no. 3, pp. 1–16.
49. Kolmogorov A.N., Petrovskii I.G., and Piskunov, N.S., Examining the Equation of a Diffusion Related to an Increase in Matter and Its Application to a Biological Problem, *Byull. Mosk. Gos. Univ. Ser. A. Mat. Mekh.*, 1937, no. 1, pp. 1–26. (In Russian.)
50. Vanag, V.K. and Epstein, I.R., Stationary and Oscillatory Localized Patterns, and Subcritical Bifurcations, *Physical Review Letters*, 2004, vol. 92, no. 12, article no. 128301.
51. Hoffmann, M, Fröhner, C, and Noé, F., ReaDDy 2: Fast and Flexible Software Framework for Interacting-Particle Reaction Dynamics, *PLOS Computational Biology*, 2019, vol. 15, no. 2, article no. e1006830.
52. Garcia-Quismondo, M., Gutiérrez-Escudero, R., Martínez-del-Amor, M.A., et al., P-Lingua 2.0: A Software Framework for Cell-Like P Systems, *International Journal of Computers, Communications & Control*, 2009, vol. 4, no. 3, pp. 234–243.
53. Dasgupta, D., Artificial Neural Networks and Artificial Immune Systems: Similarities and Differences, *Proceedings of IEEE 1997 International Conference on Computational Cybernetics and Simulation: Systems, Man, and Cybernetics*, vol. 1, Orlando, 1997, pp. 873–878.
54. Birge, B., PSOt – a Particle Swarm Optimization Toolbox for Use with Matlab, *Proceedings of the 2003 IEEE Swarm Intelligence Symposium* (*SIS'03*), Indianapolis, 2003, pp. 182–186.
55. Qi, R., Hu, B., and Cournède, P., PSOTS: A Particle Swarm Optimization Toolbox in Scilab, *Proceedings of IEEE 2009 International Workshop on Open-source Software for Scientific Computation* (*OSSC*), Guiyang, 2009, pp. 107–114.
56. Woods, D., Doty, D., Myhrvold, C., et al., Diverse and Robust Molecular Algorithms Using Repro-Grammable DNA Self-Assembly, *Nature*, 2019, vol. 567, no. 7748, pp. 366–372.
57. Hunt, J., Timmis, J., Cooke, D., et al., Jisys: Development of an Artificial Immune System for Real World Applications, in *Artificial Immune Systems and Their Applications*, Dasgupta, D., Ed., Berlin: Springer-Verlag, 1998, pp. 157–186.
58. Shen, X, Gao, X.Z., Bie, R, and Jin, X., Artificial Immune Networks: Models and Applications, *Proceedings of the International Conference on Computational Intelligence and Security* (*ICCIAS 2006*), Guangzhou, 2006, pp. 394–397.
59. Twycross, J. amd Aickelin, U., Libtissue - Implementing Innate Immunity, *Proceedings of IEEE 2006 International Conference on Evolutionary Computation* (*CEC*), Vancouver, 2006, pp. 499–506.
60. Coore, D., Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computing, *PhD thesis*, MIT, 1999.
61. Beal, J. and Bachrach, J., Infrastructure for Engineered Emergence on Sensor/Actuator Networks, *IEEE Intelligent Systems*, 2006, vol. 21, no. 2, pp. 10–19.
62. Secco, J., Farina, M., Demarchi, D., et al., Memristor Cellular Automata for Image Pattern Recognition and Clinical Applications, *Proceedings of IEEE 2016 International Symposium on Circuits and Systems* (*ISCAS*), Montreal, 2016, pp. 1378–1381.
63. Miranda, G., Machicao, J., and Bruno, O., Exploring Spatiotemporal Dynamics of Cellular Automata for Pattern Recognition in Networks, *Scientific Reports*, 2016, vol. 6, pp. 37329–37329-15.
64. Partanen, J., An Urban Cellular Automata Model for Simulating Dynamic States on a Local Scale, *Entropy*, 2017, vol. 19, no. 1, article no. 12.
65. Das, A.K. and Chattaraj, U., Heterogeneous Traffic Simulation for Urban Streets Using Cellular Automata, *Arabian Journal for Science and Engineering*, 2019, vol. 44, no. 10, pp. 8557–8571.
66. Sree, P.K. and Babu, I.R., Towards a Cellular Automata Based Network Intrusion Detection System with Power Level Metric in Wireless Adhoc Networks (IDFADNWCA), *Proceedings of*

*the 2008 International Conference on Advanced Computer Theory and Engineering* (*ICACTE*), Phuket, 2008, pp. 1071–1075.
67. Nisha, V.M. and Jeganathan, L., A Symmetry Based Anomaly Detection in Brain Using Cellular Automata for Computer Aided Diagnosis, *Indonesian Journal of Electrical Engineering and Computer Science*, 2019, vol. 14, no. 1, pp. 471–477.
68. Kabli, F., Hamou, R.M., and Amine, A., DNA Data Clustering by Combination of 3D Cellular Automata and N-Grams for Structure Molecule Prediction, *International Journal of Bioinformatics Research and Applications*, 2016, vol. 12, no. 4, pp. 299–311.
69. Berberoğlu, S., Akın, A., and Clarke, K.C., Cellular Automata Modeling Approaches to Forecast Urban Growth for Adana, Turkey: A Comparative Approach, *Landscape and Urban Planning*, 2016, vol. 153, pp. 11–27.
70. Shen, L., Shen, Y., Song, C., et al., A Novel Power System Anomaly Data Identification Method Based on Neural Network and Affine Propagation, *Proceedings of the International Conference on Artificial Intelligence and Security* (*ICAIS*), New York, 2019, pp. 499–508.
71. Ishitaki, T., Oda, T., and Barolli, L., A Neural Network Based User Identification for Tor Networks: Data Analysis Using Friedman Test, *Proceedings of the IEEE 2016 30th International Conference on Advanced Information Networking and Applications Workshops* (*WAINA*), Crans-Montana, 2016, pp. 7–13.
72. Tobiyama, S., Yamaguchi, Y., Shimada, H., et al., Malware Detection with Deep Neural Network Using Process Behavior, *Proceedings of the IEEE 40th Annual Computer Software and Applications Conference* (*COMPSAC*), vol. 2, Atlanta, 2016, pp. 577–582.
73. Manoj, K. and Charul, B., Hybrid Tracking Model and GSLM Based Neural Network for Crowd Behavior Recognition, *Journal of Central South University*, 2017, vol. 24, no. 9, pp. 2071–2081.
74. Raman, M.G., Somu, N., and Mathur, A.P., Anomaly Detection in Critical Infrastructure Using Probabilistic Neural Network, *Proceedings of the International Conference on Applications and Techniques in Information Security*, Tamil Nadu, 2019, pp. 129–141.
75. Yuan, F., Cao, Y., and Shang, Y., Insider Threat Detection with Deep Neural Network, *Proceedings of the International Conference on Computational Science*, Wuxi, 2018, pp. 43–54.
76. Jiang, Y., Li, C.H, Yu, L.S., and Bao, B., On Network Security Situation Prediction Based on RBF Neural Network, *Proceedings of the IEEE 2017 36th Chinese Control Conference* (*CCC*), Liaoning, 2017, pp. 4060–4063.
77. Pang, Y., Xue, X., and Wang, H., Predicting Vulnerable Software Components through Deep Neural Network, *Proceedings of the 2017 International Conference on Deep Learning Technologies*, Chengdu, 2017, pp. 6–10.
78. Poteralski, A., Hybrid Artificial Immune Strategy in Identification and Optimization of Mechanical Systems, *Journal of Computational Science*, 2017, vol. 23, pp. 216–225.
79. Lima, F.P, Chavarette, F.R, Souza, S.S, and Lopes, M.L., Monitoring and Fault Identification in Aeronautical Structures Using an Wavelet-Artificial Immune System Algorithm, in *Probabilistic Prognostics and Health Management of Energy Systems*, Ekwaro-Osire, S., Gonçalves, A., and Alemayehu, F., Eds., Cham: Springer, 2017, pp. 203–219.
80. Purbasari, A., Supriana, I., Santoso, O.S., and Mandala, R., Designing Artificial Immune System Based on Clonal Selection: Using Agent-Based Modeling Approach, *Proceedings of the IEEE 2013 7th Asia Modelling Symposium*, Hong Kong, 2013, pp. 11–15.
81. Liu, Y., Ding, Y., Hao, K., and Chen, L., An Immune System-Inspired Information Diffusion Model, *Proceedings of the IEEE 2017 36th Chinese Control Conference* (*CCC*), Liaoning, 2017, pp. 11238–11243.
82. Vasilyev, V. and Shamsutdinov, R., Distributed Intelligent System of Network Traffic Anomaly Detection Based on Artificial Immune System, *Proceedings of the 7th Scientific Conference on Information Technologies for Intelligent Decision Making Support* (*ITIDS 2019*), Ufa: Atlantis Press, 2019, pp. 40–45.
83. Jiang, Q. and Chang, F., A Novel Antibody Population Optimization Based Artificial Immune System for Rotating Equipment Anomaly Detection, *Journal of Mechanical Science and Technology*, 2020, vol. 34, no. 9, pp. 3565–3574.
84. Wang, M., Ge, J., Zhang, D., and Zhang, F., An Improved Artificial Immune System Model for Link Prediction, *Proceedings of the Pacific Rim International Conference on Artificial Intelligence* (*PRICAI*), Nanjing, 2018, pp. 1–9.
85. Günay, M., Orman, Z., Ensari, T., et al., Diagnosis of Lung Cancer Using Artificial Immune System, *Proceedings of IEEE 2019 Scientific Meeting on Electrical-Electronics & Biomedical Engineering and Computer Science* (*EBBT*), Istanbul, 2019, pp. 1–4.
86. Maji, P., Shaw, C., Ganguly, N., et al., Theory and Application of Cellular Automata for Pattern Classification, *Fundamenta Informaticae*, 2003, vol. 58, no. 3–4, pp. 321–354.
87. Zhou, L. and Yang, M., A Classifier Build around Cellular Automata for Distributed Data Mining, *Proceedings of the IEEE 2008 International Conference on Computer Science and Software Engineering*, vol. 4, Wuhan, 2008, pp. 312–315.
88. Chang, H., Baek, S., Kim, H., et al., Development of Distributed Real-Time Decision Support System for Traffic Management Centers Using Microscopic CA Model, *Iranian Journal of Science & Technology, Transaction B, Engineering*, 2007, vol. 31, no. B2, pp. 155–166.
89. Benhacine, F.Z., Atmani, B., Benamina, M., et al., A Visual Decision Making Support System for the Diabetes Prevention, in *Advances in Data Science, Cyber Security and IT Applications*, Alfaries, A., Mengash, H., Yasar, A., and Shakshuki, E., Eds., *Proceedings of the 1st International Conference on Computing* (*ICC 2019*), Part II, Riyadh, 2019, pp. 81–92.
90. Liu, Y. and He, J., Developing a Web-Based Cellular Automata Model for Urban Growth Simulation, *Proceedings of the International Symposium on Spatial Analysis, Spatial-Temporal Data Modeling, and Data Mining*, SPIE, vol. 7492, Wuhan, 2009, article no. 74925C.
91. Wainer, G., Developing a Software Toolkit for Urban Traffic Modeling, *Software: Practice and Experience*, 2007, vol. 37, no. 13, pp. 1377–1404.
92. Li, Y., Zhang, H., and Shen, Q., Spectral–Spatial Classification of Hyperspectral Imagery with 3D Convolutional Neural Network, *Remote Sensing*, 2017, vol. 9, no. 1, pp. 67– 88.
93. Jia, F., Lei, Y., Lu, N., and Xing, S., Deep Normalized Convolutional Neural Network for Imbalanced Fault Classification of Machinery and Its Understanding via Visualization, *Mechanical Systems and Signal Processing*, 2018, vol. 110, pp. 249–367.
94. Beşikçi, E.B., Arslan, O., Turan, O., and Ölçer, A.I., An Artificial Neural Network Based Decision Support System for Energy Efficient Ship Operations, *Computers & Operations Research*, 2016, vol. 66, pp. 393–401.
95. Boonpeng, S. and Jeatrakul, P., Decision Support System for Investing in Stock Market by Using OAA-Neural Network, *Proceedings of the IEEE 2016 8th International Conference on Advanced Computational Intelligence* (*ICACI*), Chiang Mai, 2016, pp. 1–6.

96. Tang, G. and Zeng, H., Chemical Production Information Management System Based on Artificial Intelligence Neural Network Algorithm, *Chemical Engineering Transactions*, 2018, vol. 66, pp. 967–972.
97. Elechi, P., Improved Ghost Worker Fraud Detection System Using Artificial Neural Network, *Journal of Electrical Engineering, Electronics, Control and Computer Science*, 2019, vol. 5, no. 1, pp. 17–24.
98. Magna, G., Casti, P., Jayaraman, S.V., et al., Identification of Mammography Anomalies for Breast Cancer Detection by an Ensemble of Classification Models Based on Artificial Immune System, *Knowledge-Based Systems*, 2016, vol. 101, pp. 60–70.
99. Aldhaheri, S., Alghazzawi, D., Cheng, L., et al., DeepDCA: Novel Network-Based Detection of IoT Attacks Using Artificial Immune System, *Applied Sciences*, 2020, vol. 10, no. 6, pp. 1909–1909-23.
100. Mnif, S., Elkosantini, S., Darmoul, S., and Said, L.B., An Immune Multi-agent Based Decision Support System for the Control of Public Transportation Systems, *Proceedings of the International Conference on Practical Applications of Agents and Multi-Agent Systems*, Seville, 2016, pp. 187–198.
101. Berquedich, M., Kamach, O., Masmoudi, M., and Deshayes, L., Agile Decision Support System for the Management of Tensions in Emergency Services Using AIS Techniques, *Proceedings of the 2017 IEEE International Colloquium on Logistics and Supply Chain Management* (*LOGISTIQUA*), Rabat, 2017, pp. 118–123.
102. do Nascimento Alves, H., Machado, R.C., and Bergê, I.G., Design and Development of a Software for Fault Diagnosis in Radial Distribution Networks, *Proceedings of the IEEE/IAS 9th International Conference on Industry Applications* (*INDUSCON 2010*), São Paulo, 2010, pp. 1–6.
103. Chitra, M.E. and Rajaram, M., A Software Reliability Estimation Tool Using Artificial Immune Recognition System, *Proceedings of the International MultiConference of Engineers and Computer Scientists* (*IMECS 2008*), vol. 1, Hong Kong, 2008, pp. 967–975.

*This paper was recommended for publication by V.V. Kul'ba, a member of the Editorial Board.*

*Received January 28, 2021, and revised March 31, 2021.*
*Accepted April 6, 2021.*

**Author information**

**Shiroky, Aleksandr Aleksandrovich.** Cand. Sci. (Phys.–Math.), Trapeznikov Institute of Control Sciences, Russian Academy of Sciences, Moscow, Russia, ✉ shiroky@ipu.ru

**Kalashnikov, Andrei Olegovich.** Dr. Sci. (Eng.), Trapeznikov Institute of Control Sciences, Russian Academy of Sciences, Moscow, Russia, ✉ aokalash@ipu.ru

**Cite this article**

Shiroky, A.A., Kalashnikov, A.O., Natural Computing with Application to Risk Management in Complex Systems. *Control Sciences* **4,** 2–17 (2021). http://doi.org/10.25728/cs.2021.4.1

Original Russian Text © Shiroky, A.A., Kalashnikov, A.O., 2021, published in *Problemy Upravleniya*, 2021, no. 4, pp. 3–20.