Available online at www.sciencedirect.com
ScienceDirect
Physics of Life Reviews 4 (2007) 1–36
PHYSICS of LIFE reviews
www.elsevier.com/locate/plrev

Review

Fundamentals of natural computing: an overview

Leandro Nunes de Castro *

Graduate Program in Computer Science, Catholic University of Santos, R. Dr. Carvalho de Mendonça, 144, Vila Mathias, Santos SP, Brazil

Received 7 October 2006; accepted 9 October 2006

Available online 4 December 2006

Communicated by L. Perlovsky

Abstract

Natural computing is a terminology introduced to encompass three classes of methods: (1) those that take inspiration from nature for the development of novel problem-solving techniques; (2) those that are based on the use of computers to synthesize natural phenomena; and (3) those that employ natural materials (e.g., molecules) to compute. The main fields of research that compose these three branches are the artificial neural networks, evolutionary algorithms, swarm intelligence, artificial immune systems, fractal geometry, artificial life, DNA computing, and quantum computing, among others. This paper provides an overview of the fundamentals of natural computing, particularly the fields listed above, emphasizing the biological motivation, some design principles, their scope of applications, current research trends and open problems. The presentation is concluded with a discussion about natural computing, and when it should be used.

Keywords: Natural computing; Bio-inspired computing; Problem-solving; Novel computing paradigms

Contents

* Tel.: +55 13 3251 2867; fax: +55 13 3226 0500.
E-mail address: lnunes@unisantos.br (L.N. de Castro).

1571-0645/$ – see front matter
doi:10.1016/j.plrev.2006.10.002

1. Introduction

Most computer users and programmers are accostumed with a static machine that can only do what you tell it to do. The computers have demonstrated to be very useful in many domains and applications, from the storage of information in a certain company to the complex decision-making process of landing an aircraft. Although not transparent to most of these end users, this capability of performing tasks, solving complex problems and simulating things is an outcome of decades of research. These decades of research have also witnessed the emergence and testing of a number of different computational paradigms and the use of computers as a means of making other methods more efficient. For instance, the whole field of mathematical programming, which started by the mid 1940s with the works on linear programming, greatly benefitted from the development and improvement of computers.

Another scientific trend that came onto the scene by the mid 1940s, but which received a great deal of attention over the past two or three decades, is the merging of ideas from nature and computing, words that used to mean antagonist things. Researchers are currently using ideas from nature to develop novel problem-solving techniques; computers

have been used to synthesize natural phenomena; and novel natural materials (in addition to silicon) have been used to perform computation. These three distinct, but inter-related approaches, altogether constitute what is now known as *natural computing* [69,70].

The present paper provides an introductory tour of these three natural computing branches (computing inspired by nature; synthesis of natural phenomena in computers; and computing with natural materials) briefly describing the biological motivation, main players, design principles, scope of each branch, current trends and open problems. Section 2 presents natural computing and its three main branches as a new discipline. Section 3 introduces nature-inspired computing, whilst Section 4 provides an overview about the synthesis of natural phenomena by computers. Section 5 presents computing with natural materials, and Section 6 discusses when natural computing should be applied. The paper is concluded in Section 7.

2. Natural computing

Natural computing is the computational version of the process of extracting ideas from nature to develop computational systems, or using natural materials (e.g., molecules) to perform computation. It can be divided into three main branches [69,76]:

(1) *Computing inspired by nature*: it makes use of nature as inspiration for the development of problem solving techniques. The main idea of this branch is to develop computational tools (algorithms) by taking inspiration from nature for the solution of complex problems.
(2) *The simulation and emulation of nature by means of computing*: it is basically a synthetic process aimed at creating patterns, forms, behaviors, and organisms that (do not necessarily) resemble 'life-as-we-know-it'. Its products can be used to mimic various natural phenomena, thus increasing our understanding of nature and insights about computer models.
(3) *Computing with natural materials*: it corresponds to the use of novel natural materials to perform computation, thus constituting a true novel computing paradigm that comes to substitute or supplement the current silicon-based computers.

Therefore, natural computing can be defined as the field of research that, based on or inspired by nature, allows the development of new computational tools (in software, hardware or 'wetware') for problem solving, leads to the synthesis of natural patterns, behaviors, and organisms, and may result in the design of novel computing systems that use natural media to compute.

Natural computing is thus a field of research that testimonies against the specialization of disciplines in science. It shows, with its three main areas of investigation, that knowledge from various fields of research are necessary for a better understanding of life, for the study and simulation of natural systems and processes, and for the proposal of novel computing paradigms. Physicists, chemists, engineers, biologists, computer scientists, among others, all have to act together or at least share ideas and knowledge in order to make natural computing feasible.

Most of the computational approaches natural computing deals with are based on highly simplified versions of the mechanisms and processes present in the corresponding natural phenomena. The reasons for such simplifications and abstractions are manifold. First of all, most simplifications are necessary to make the computation with a large number of entities tractable. Also, it can be advantageous to highlight the minimal features necessary to enable some particular aspects of a system to be reproduced and to observe some emergent properties. Which level is most appropriate for the investigation and abstraction depends on the scientific question asked, what type of problem one wants to solve, or the life phenomenon to be synthesized.

Natural computing usually integrates experimental and theoretical biology, physics and chemistry, empirical observations from nature and several other sciences, facts and processes from different levels of investigation into nature so as to achieve its goals, as summarized in Fig. 1.

3. Computing inspired by nature

Among all natural computing approaches, computational algorithms and systems inspired by nature are the oldest and most popular ones. They arose with two main objectives in mind. First, researchers were interested in the

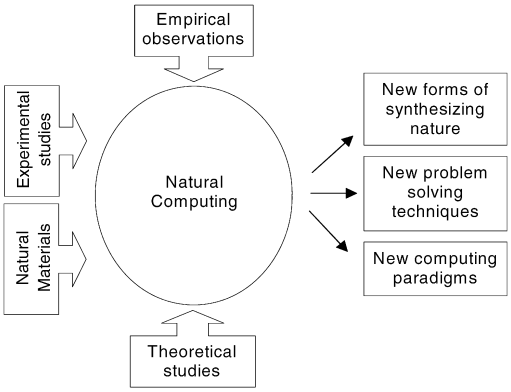

Fig. 1. Many approaches are used to develop natural computing and its main branches.

modeling of natural phenomena and their simulation in computers. The common goal in this direction is to devise theoretical models, which can be implemented in computers, faithful enough to the natural mechanisms investigated so as to reproduce qualitatively and/or quantitatively some of their functioning. The second objective involves the study of natural phenomena, processes and even theoretical models for the development of computational systems and algorithms capable of solving complex problems. The motivation, in this case, is to provide (alternative) solution techniques to problems that could not be (satisfactorily) resolved by other more traditional techniques, such as linear, non-linear, and dynamic programming. In such cases, the computational techniques developed can also be termed *bio-inspired computing* or *biologically motivated computing* [76,186], or *computing with biological metaphors* [218]. Among the many approaches within computing inspired by nature, the most well-known ones are the artificial neural networks, evolutionary algorithms, swarm intelligence and artificial immune systems.

3.1. Artificial neural networks

A landmark work in the branch of nature-inspired computing was the paper by MC Culloch and Pitts [192], which introduced the first mathematical model of a neuron that gave rise to the *artificial neural networks*—ANNs [32,97, 134,167]. ANNs can be defined as information processing systems designed with inspiration taken from the nervous system, in most cases the human brain. Currently, most works on ANNs place particular emphasis on problem solving. In this sense, ANNs are distinct from what is currently known as *computational neuroscience* [66,209,270], which is mainly concerned with the development of biological models of the nervous system.

3.1.1. Biological motivation

Neurons are the basic units used for computation in the brain, and their simplified abstract models are the basic processing units of ANNs. Neurons are connected to one another by a small junction called synapse, whose capability of being modulated is believed to be the basis for most of our cognitive abilities, such as perception, thinking, and inferring. Neurons can have *forward* and *feedback* connections to other neurons, meaning that they can have either one way or reciprocal connections with other neurons in the nervous system. These interconnected neurons give rise to what is known as *networks of neurons* or *neural networks*. A small number of interconnected neurons (units) can exhibit complex behaviors and information processing capabilities that cannot be observed in single neurons [49].

One important feature of neural networks is the representation of information (knowledge) in a *distributed* way, and the *parallel processing* of this information. The nervous system is continuously modifying and updating itself. Virtually all its functions, including perception, motor control, thermoregulation, and reasoning, are modifiable by experience. The topography of the modifications appears not to be final and finished, but an ongoing process with a virtually interminable schedule. A broad range of structural adaptability can be conveniently condensed by referring simply to synapses, since every modification either involves synaptic modification directly or indirectly, or can be reasonably so represented. Learning by means of setting synaptic efficiency is thus the most important mechanism in neural networks, biological and artificial.

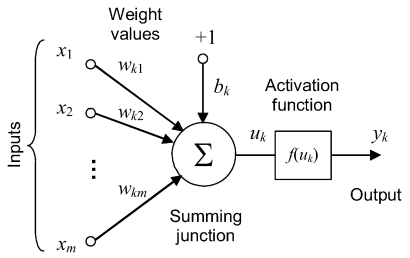

Fig. 2. Nonlinear model of a neuron.

3.1.2. Design principles

Artificial neural networks (ANN) present a number of features and performance characteristics in common with the nervous system [134]:

- The basic information processing occurs in many simple elements called (artificial) *neurons*, *nodes* or *units*.
- These neurons can *receive* and *send stimuli* from and to other neurons and the environment.
- Neurons can be connected to one another forming *neural networks*.
- Information (signals) are transmitted between neurons via connection links called *synapses*.
- The efficiency of a synapse, represented by an associated *strength* or *weight value*, corresponds to the information stored in the neuron, thus in the network.
- Knowledge is acquired from the environment by a process known as *learning*, which is basically responsible for *adapting* the connection strengths (weight values) to the environmental stimuli.

From a design perspective, an ANN can be characterized by three main features: (1) a set of *artificial neurons*, also termed *nodes*, *units*, or simply *neurons*; (2) the pattern of connectivity among neurons, called the network *architecture* or *structure*; and (3) a method to determine the weight values, called its *training* or *learning* algorithm. Although there are several neuron models, network architectures and learning algorithm, this paper reviews only the most standard types. Some comments concerning more recent models, architectures and algorithms will be provided in a further section.

Artificial neurons In the biological neuron, the input signals come into the cell via channels located in synapses, allowing ions to flow into and out of the neuron. A membrane potential appears as a result of the integration of the neural inputs, and will then determine whether a given neuron will produce a spike (action potential) or not. This spike causes neurotransmitters to be released at the end of the axon, which then forms synapses with the dendrites of other neurons. An action potential occurs when the membrane potential is above a critical threshold level. The net effect of these biological processes is summarized in the computational models discussed here by a *weight* between two neurons. The computing element employed in most neural networks is an integrator and computes based on its connection strengths. Fig. 2 illustrates a standard artificial neuron, depicting its most important parts: the synapses, characterized by their *weight values*; the *summing junction* (*integrator*); and the *activation function* $f(u_k)$.

Specifically, an input signal x_j at the input of synapse j connected to neuron k is multiplied by the synaptic weight w_{kj}. The summing junction adds all input signals weighted by the synaptic weight values plus the neuron's bias b_k; this operation constitutes the dot (or inner) product between the inputs and the weight values, plus the bias b_k. Finally, an activation function is used to limit the amplitude of the output of the neuron. Mathematically, the output, y_k, of neuron k can be described by a simple equation:

$$y_k = f(u_k) = f\left(\sum_{j=1}^{m} w_{kj}x_j + b_k\right) \tag{1}$$

where x_j, $j = 1, \ldots, m$, are the input signals; w_{kj}, $j = 1, \ldots, m$, are the synaptic weights of neuron k; u_k is the net input to the activation function; b_k is the neuron bias; $f(\cdot)$ is the activation function; and y_k is the output.

The activation function, denoted by $f(u_k)$, determines the output of a neuron k in relation to its net input u_k. It can thus assume a number of forms, of which some of the most frequently used are the linear function, the threshold

function, the radial basis functions (e.g., Gaussian function), and the sigmoid functions (e.g., logistic and hyperbolic tangent).

Network architectures In ANNs, a layer of neurons refers to functional layers of nodes. Most ANNs employ some standardized architectures, specially designed for the engineering purposes of solving problems. There are basically three possible layers in a network: an *input layer*, one or more *intermediate* (or *hidden*) *layers*, and an *output layer*. In general, it is possible to distinguish three main types of network architectures: *single-layer feedforward networks*, *multi-layer feedforward networks*, and *recurrent networks*.

The simplest case of layered networks consists of an input layer of nodes whose output feeds the output layer. Usually, the input nodes are linear. In contrast, the output units are usually processing elements. The signal is propagated in this network in a purely positive or *feedforward* manner; that is, signals are propagated from the network inputs to its outputs and never the opposite way (*backward*). The second class of feedforward neural networks is known as multi-layer networks. These are distinguished from the single-layered networks by the presence of one or more *intermediate* or *hidden* layers. By adding one or more nonlinear hidden layers, the computational processing and storage capability of the network is increased. The output of each network layer is used as input to the following layer. Finally, the third main class of networks is known as recurrent networks, distinguished from feedforward networks for they have at least one *recurrent* (or *feedback*) loop. For instance, a recurrent network may consist of a single layer of neurons with each neuron feeding its output signal back to the input of other neurons. In a feedback arrangement, there is communication between neurons, there can be cross talk and plan revision, intermediate decisions can be taken, and a mutually agreeable solution can be found.

Learning approaches In standard neural networks, *learning* (or *training*) corresponds to the process by which the network's weights are adjusted through a mechanism of presentation of input stimuli. More sophisticated learning algorithms are capable of dynamically adjusting several other parameters of an ANN, such as the network architecture and the activation function of individual neurons [174,216]. The environmental or input stimuli correspond to a set of *input data* (or *patterns*) that is used to *train* the network.

For most applications there are two steps involved in the use of the ANN: (1) training, and (2) application. With standard learning algorithms a neural network learns through an iterative process of weight adjustment. The type of learning is defined by the way in which the weights are adjusted. The three main learning approaches are: (1) *supervised learning*, (2) *unsupervised learning*, and (3) *reinforcement learning*.

Supervised learning embodies the concept of a *supervisor* or *teacher*, who has the knowledge about the environment in which the network is operating. This knowledge is represented in the form of a set of *input-output samples* or *patterns*. The network free parameters are adjusted through the combination of the input and error signals, where the error signal e_j is the difference between the desired output d_j and the current network output y_j.

$$e_j = d_j - y_j. \tag{2}$$

In the *unsupervised* or *self-organized* learning approach, there is no supervisor to evaluate the network performance in relation to the input data set. The classes of the data are *unknown* or *unlabelled* and there is no supervisor. The network adapts itself to statistical regularities in the input data, developing an ability to create internal representations that encode the features of the input data. Usually, self-organizing algorithms employ a *competitive learning* scheme, in which the network output neurons compete with one another to become activated.

Reinforcement learning is distinguished from the other approaches as it relies on learning from direct interaction with the environment, but does not rely on explicit supervision or complete models of the environment. Often, the only information available is a scalar evaluation that indicates how well the artificial neural network is performing. This is based on a framework that defines the interaction between the neural network and its environment in terms of the current values of the network's free parameters (weights), network response, and rewards. Situations are mapped into actions so as to maximize a numerical reward signal [265].

Supervised and unsupervised learning are the most popular learning paradigms in neural networks. There are several algorithms in the literature within each of these categories. The most traditional supervised neural networks are the *single-layer perceptron* [242], the *adaptive linear element* (ADALINE) [277], the *multi-layer perceptron* [245],

and the *radial basis function* network [275]. The most well-known unsupervised neural network is the Kohonen's self-organizing feature map [167].

3.1.3. Scope of artificial neural networks

Artificial neural networks can, in principle, compute any computable function, i.e., they can do everything a standard digital computer can do, and can thus be classified as a type of *neurocomputer*. Artificial neural networks are particularly useful for clustering, classification, pattern recognition, and function approximation (mapping) problems to which hard and fast rules cannot be easily applied. Vasts lists of practical and commercial applications of ANN, including links to other sources, can be found at the Frequently Asked Questions (FAQ) site for neural networks: ftp://ftp.sas.com/pub/neural/FAQ7.html#A_applications. The applications listed in this site include agriculture, automotive, chemistry, criminology, biometric recognition, finance and economics, games, sports, gambling, industry, medicine, music, robotics, signal processing, time-series prediction, and weather forecasting.

3.1.4. Current trends and open problems

Among all natural computing techniques, artificial neural networks constitute the oldest and largest field. As such, research in this area is performed in many frontlines, such as applications, theoretical investigations and improvements, hybridization with other methods, and biological modelling. The field has passed through many phases, from an early period of great excitement (up to late 1960s) to some periods of discredit (from the late 1960s to the mid 1980s). Since the publication of the Parallel Distributed Processing volumes in 1986 [191,245], ANNs have received a great deal of attention. Among the main current trends investigated it is possible to list: networks with constructive architectures, i.e., networks whose architecture may increase and/or decrease in size [144,174,180]; ensembles of networks [228,289]; the investigation into formal aspects of ANNs, such as convergence and universal approximation capabilities [119,131,281]; hybrids between ANN and other approaches (e.g., [284]); neural networks based on the statistical learning theory [271], such as support vector machines [42,56,57]); and more biologically plausible networks (e.g., [65,227]). In terms of future directions, it is possible to stress the application of biologically and cognitively inspired neural networks to complex problems, including integrated collaborative systems of multiple interactive agents, bioinformatics, signal processing, data mining, web mining, biometrics and time-series data. Among open problems there is knowledge extraction [143,149], the modeling of higher human cognitive abilities, interaction between emotional and conceptual, and the unification of language and cognition in evolving and integrated systems [227].

3.2. Evolutionary computing

Evolutionary computing, also called *evolutionary computation*, is the field of research that draws ideas from evolutionary biology in order to develop search and optimization techniques for solving complex problems [14,15,102,118, 142,194,196]. Most *evolutionary algorithms* are rooted on evolutionary biology, which basically states that a population of individuals capable of reproducing and subjected to genetic variation followed by selection results in new populations of individuals increasingly fitter to their environment. The computational abstraction of these procedures resulted in the so-called evolutionary algorithms.

3.2.1. Biological motivation

Evolutionary biology is concerned with the study of the diversity of life, the differences and similarities among organisms, and the characteristics of organisms. An evolving system corresponds to the one in which there is a descent of entities over time, one generation after the other, and in which characteristics of the entities differ across generations and endow them with some survival or reproductive advantage [110]. Therefore, *evolution* can be broadly defined as *descent with modification* and often *with diversification*. Adaptation as a result of variation plus natural selection leads to improvement in the function of an organism and its many component parts. Evolution is a result of one or more reproducing populations of individuals, which suffer genetic variation followed by selection. The variation affects the genetic makeup of individuals (genotype), which will present a selective advantage over the others if their phenotypes (physical and chemical characteristics) confer them a better adaptability to the environment they inhabit. This degree of adaptability to the environment is broadly termed fitness. The genetic entities that suffer variation are located within the cell nucleus and are named chromosomes, whose basic functional units are the genes. In both types

of reproduction, asexual and sexual, there may occur a deviation (mutation) in one or more alleles of a chromosome, allowing the appearance of a new character in the phenotype of the offspring individual. In sexual reproduction, in addition to mutation, there is the recombination (crossing-over) of parental genetic material, resulting in offspring that present features in common with both parents. The survival and reproductive advantage of an individual organism, its fitness, endows it with a selective advantage over the others, less fit individuals.

3.2.2. Design principles

The basic idea of EC, which came onto the scene about the 1950s to 1960s, has been to make use of the powerful process of natural evolution as a problem-solving paradigm, usually by simulating it on a computer. The original three mainstreams of EC are *genetic algorithms* (GAs) [118,196], *evolution strategies* (ES) [29,247], and *evolutionary programming* (EP) [14,15,102]. Another mainstream of evolutionary computation that has been receiving increasingly more attention is *genetic programming* [169,170]. Despite some differences among these approaches, all of them present the basic features of an evolutionary process. A standard evolutionary algorithm can thus be proposed as follows:

- *A population of individuals that reproduce with inheritance*. Each individual represents or encodes a point in a search space of potential solutions to a problem. These individuals are allowed to reproduce (sexually or asexually), generating offspring that inherit some traits from their parents. These inherited traits cause the offspring to present resemblance with their progenitors.
- *Genetic variation*. Offspring are prone to genetic variation through mutation, which alters their genetic makeup. Mutation allows the appearance of new traits in the offspring and, thus, the exploration of new regions of the search space.
- *Natural selection*. The evaluation of individuals in their environment results in a measure of adaptability, or fitness value to be assigned to them. A comparison of individual fitnesses will lead to a competition for survival and reproduction in the environment, and there will be a selective advantage for those individuals with higher fitness.

The standard evolutionary algorithm is a generic, iterative, and probabilistic algorithm that maintains a population $\mathbf{P}$ of N individuals, $\mathbf{P} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, at each iteration. Each individual, represented as a *data structure*, corresponds to (encodes) a potential solution to a problem that has to be solved. The individuals $\mathbf{x}_i$, $i = 1, \ldots, N$, are evaluated to give their measures of adaptability to the environment, or *fitness*. Then, a new population, at iteration $t + 1$, is generated by *selecting* some (usually the most fit) individuals from the current population and *reproducing* them, sexually or asexually. If employing sexual reproduction, a *genetic recombination* (*crossover*) operator may be used. Genetic variations through *mutation* may also affect some individuals of the population, and the process iterates. The completion of all these steps: reproduction, genetic variation, and selection, constitutes what is called a *generation*.

Due to its popularity and space constraints, this review will only discourse about the *genetic algorithms* (GAs). GAs are those evolutionary algorithms that use a vocabulary borrowed from natural genetics. This method has been offered by a number of researchers (e.g., [12,40,106,109]), but its most popular version, known as the canonical or standard GA, was presented by J. Holland [142]. This is the version to be reviewed here.

The data structures representing the *individuals* (*genotypes*) of the population are often called *chromosomes*; these are one-chromosome individuals. In standard GAs the individuals are represented as *bitstrings*. Each unit of a chromosome is a *gene*, located in a certain place in the chromosome called *locus*. The different forms a gene can assume are the *alleles*. The problem to be solved is defined and captured in an *objective function* that allows to evaluate the *fitness* of any potential solution. Each genotype, in this case a single chromosome, represents a potential solution to a problem. The meaning of a particular chromosome, its *phenotype*, is defined according to the problem under study.

As each chromosome $\mathbf{x}_i$, $i, \ldots, N$, often corresponds to the encoded value of a candidate solution, it often has to be decoded into a form appropriate for evaluation and is then assigned a fitness value according to the objective. Each chromosome is assigned a probability of reproduction, p_i, $i, \ldots, N$, so that its likelihood of being selected is proportional to its fitness relative to the other chromosomes in the population. If the fitness of each chromosome is a strictly positive number to be maximized, selection is traditionally performed via an algorithm called *fitness proportional selection* or *Roulette Wheel selection* [100].

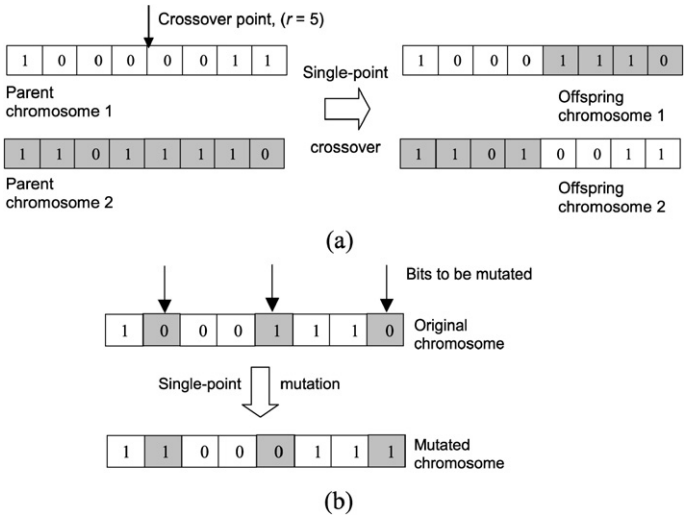

Fig. 3. Crossover and mutation. (a) Single-point crossover for a pair of chromosomes of length $l = 8$. (b) Point mutations for a chromosome of length $l = 8$ (three points are selected for being mutated).

The assigned probabilities of reproduction result in the generation of a population of chromosomes probabilistically selected from the current population. The selected chromosomes will generate offspring via the use of specific *genetic operators*, such as *crossover*, and *mutation* might be used to introduce genetic variation in the individuals, as illustrated in Fig. 3.

3.2.3. Scope of evolutionary computing

Beasley [21] divided the possible application areas of evolutionary algorithms into five broad categories: planning (e.g., routing, scheduling and packing); design (e.g., signal processing); simulation and identification; control (general plant control); and classification (e.g., machine learning, pattern recognition and classification). More specifically, evolutionary algorithms have been applied in fields like art and music composition [27,28], electronics [286], language [208], robotics [205], engineering [63], data mining and knowledge discovery [107], industry [155], signal processing [101], and many others.

3.2.4. Current trends and open problems

Although the evolutionary computation roots date from the late 1950s to the early 1960s, only from the 1990s onwards the field of evolutionary computation suffered a major growth. The main current trends in the field include theoretical investigations (e.g., [60,140]); applications to dynamic, multi-modal, multi-objective and constrained optimization (e.g., [13,94,273]); investigations into evolutionary robotics, evolutionary hardware and evolutionary electronics [175,205,249,286]; biological applications and bioinformatics [17,103]; applications in data mining, games, arts and music [55,173,283]; and many others. Open problems and main challenges include the design of high-performance evolutionary algorithms, the incorporation of domain knowledge into the algorithms, the development of on-line evolutionary systems in which adaptation and evolution occur simultaneously, the identification of the main classes of problems to which evolutionary algorithms are the most suitable approaches, and the understanding of why and how they work or do not work [285].

3.3. Swarm intelligence

The term *swarm intelligence* (SI) was coined in the late 1980s to refer to cellular robotic systems in which a collection of simple agents in an environment interact according to local rules [24,25]. Some definitions of swarm intelligence can be found in the literature:

"Swarm intelligence is a property of systems of unintelligent agents of limited individual capabilities exhibiting collectively intelligent behavior." [276, p. 333]

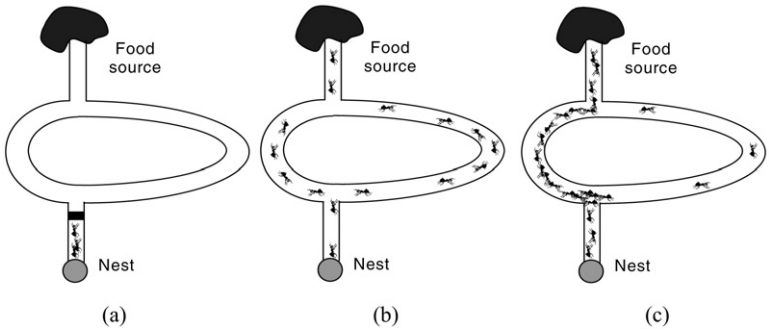

Fig. 4. An experimental set up that can be used to demonstrate that the ant *I. Humilis* is capable of finding the shortest path between the nest and a food source. (a) The bridge is initially closed. (b) Initial distribution of ants after the bridge is open. (c) Distribution of ants after some time has passed since they were allowed to exploit the food source.

"[Swarm Intelligence] include[s] any attempt to design algorithms or distributed problem-solving devices inspired by the collective behavior of social insects and other animal societies." [38, p. 7]

Two main lines of research can be identified within swarm intelligence: (1) the works based on social insects; and (2) the works based on the ability of human societies to process knowledge. Although the resultant approaches are quite different in sequence of steps and sources of inspiration, they present some commonalities. In general terms, both of them rely upon a population (colony or swarm) of individuals (social insects or particles) capable of interacting (directly or indirectly) with the environment and one another. This section starts with a discussion of social insects with particular emphasis on the foraging behavior of ants. The other approach reviewed here is the so-called particle swarm (PS) optimization algorithm, based on sociocognition [159].

3.3.1. Ant colony optimization: biological motivation

Goss et al. [121] and Deneubourg [78] performed a number of experiments and demonstrated that the trail-laying trail-following behavior of some ant species enables them to find the shortest path between a food source and the nest. They used a simple yet elegant experimental set up with the Argentine ant *Iridomyrmex humilis*. Basically, their experiment can be summarized as follows. Laboratory colonies of *I. humilis* were given access to a food source in an arena linked to the nest by a bridge consisting of two branches of different lengths, arranged so that a forager going in either direction (from the nest to the food source or vice-versa) must choose between one or the other branch, as illustrated in Fig. 4. One experimental observation is that, after a transitory phase that can last a few minutes, most of the ants choose the shortest path. It is also observed that an ant's probability of selecting the shortest path increases with the difference in length between the two branches. The explanation for this capability of selecting the shortest route from the nest to the food source comes from the use of pheromone as an indirect form of communication, known as *stigmergy*, mediated by local modifications of the environment. The problem of finding the shortest route from the nest to the food source is akin to the well-known traveling salesman problem (TSP). The salesman has to find the shortest route by which to visit a given number of cities, each exactly once. The goal is to minimize the cost (distance) of travel.

Inspired by the experiments of Goss et al. [121,122], Dorigo and collaborators [86] extended this ant model to solve the traveling salesman problem. Their approach relies on *artificial ants* laying and following *artificial pheromone trails*. While traversing a path from one city to another an ant deposits some pheromone on it. The amount of pheromone being inversely proportional to the overall length of the tour: the shorter the tour length, the more pheromone released; and vice-versa. After all the artificial ants have completed their tours and released pheromone, the links belonging to the highest number of shorter tours will have more pheromone deposited. Because the pheromone evaporates with time, links in longer routes will eventually contain much less pheromone than links in shorter tours. As the colony of artificial ants is allowed to travel through the cities a number of times, those tours less reinforced (by pheromone) will attract fewer ants in their next travel [36]. Dorigo and collaborators [86] have found that by repeating this process a number of times, the artificial ants are able to determine progressively shorter routes.

3.3.2. Ant colony optimization: basic algorithm

Ant colony optimization (ACO) algorithms constitute all algorithms for discrete optimization that took inspiration from the observation of the foraging behavior of ant colonies [85]. Although some authors have already developed versions of ant algorithms for continuous optimization (e.g., [30]), not much research has been conducted in this direction and, thus, the focus of the discussion to be presented here is on ACO for discrete optimization.

Assuming a connected graph $G = (V, E)$, the simple ACO algorithm (S-ACO) can be used to find a solution to the shortest path problem defined on the graph G. A solution is a path on the graph connecting a source node s to a destination node d and the path length is given by the number of edges traversed [83]. Associated with each edge (i, j) of the graph there is a variable τ_{ij} termed *artificial pheromone trail*, or simply pheromone. Every *artificial ant* is capable of 'marking' an edge with pheromone and 'smelling' (reading) the pheromone on the trail.

Each ant traverses one node per iteration step t and, at each node, the local information about its pheromone level, τ_{ij}, is used by the ant such that it can probabilistically decide the next node to move to, according to the following rule:

$$p_{ij}^{k}(t) = \begin{cases} \frac{\tau_{ij}(t)}{\sum_{j \in N_i} \tau_{ij}(t)} & \text{if } j \in N_i, \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where $p_{ij}^{k}(t)$ is the probability that ant k located at node i moves to node j, $\tau_{ij}(t)$ is the pheromone level of edge (i, j), all taken at iteration t, and N_i is the set of one step neighbors of node i.

While traversing an edge (i, j), the ant deposits some pheromone on it, and the pheromone level of edge (i, j) is updated according to the following rule:

$$\tau_{ij}(t) \leftarrow (1 - \rho)\tau_{ij}(t) + \Delta\tau \tag{4}$$

where t is the iteration counter, $\Delta\tau$ is the constant amount of pheromone deposited by the ant, and $\rho \in (0, 1]$ is the pheromone decay rate.

An ACO algorithm alternates, for a maximum number of iterations, the application of two basic procedures [38,87]. Assume a search is being performed on a connected graph $G = (V, E)$ with V nodes and E edges:

(1) A parallel solution construction/modification procedure in which a set of N ants builds/modifies in parallel N solutions to the problem under study.
(2) A pheromone trail updating procedure by which the amount of pheromone trail on the problem graph edges is changed.

The process of building or modifying a solution is made in a probabilistic fashion, and the probability of a new edge to be added to the solution being built is a function of the edge *heuristic desirability* η, and of the *pheromone trail* τ deposited by previous ants. The heuristic desirability η expresses the likelihood of an ant moving to a given edge. For instance, in cases the minimal path is being sought among a number of edges, the desirability η is usually chosen to be the inverse of the distance between a pair of nodes.

3.3.3. Particle swarm optimization: biological motivation

The *particle swarm* (PS) algorithm has as one of its motivations to create a simulation of human social behavior; that is, the ability of human societies to process knowledge [158–160]. PS takes into account a population of individuals capable of interacting with the environment and one another, in particular some of its neighbors. Thus, population level behaviors will emerge from individual interactions. Although the original approach has also been inspired by particle systems and the collective behavior of some animal societies, the main focus of the algorithm is on its social adaptation of knowledge; the same focus taken here.

In the PS algorithm, individuals searching for solutions to a given problem learn from their own past experience and from the experiences of others. Individuals evaluate themselves, compare to their neighbors and imitate only those neighbors who are superior to themselves. Therefore, individuals are able to evaluate, compare and imitate a number of possible situations the environment offers them. The most typical PS algorithm searches for optima in an L-dimensional real-valued space, $\Re^L$. Thus, the variables of a function to be optimized can be conceptualized as a vector that corresponds to a point in a multidimensional search space. Multiple individuals can thus be plotted within a single set of coordinates, where a number of individuals will correspond to a set of points or particles in the space.

3.3.4. Particle swarm optimization: basic algorithm

In mathematical terms, the PS algorithm can be implemented as follows. The position of a particle i is given by $\mathbf{x}_i$, which is an L-dimensional vector in $\Re^L$. The change in position of a particle is its velocity, denoted by $\mathbf{v}_i$:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1). \tag{5}$$

The particle swarm algorithm samples the search-space by modifying the velocity of each particle. The question that remains to be answered is how to define the velocity of the particle so that it moves in the appropriate directions and with the appropriate 'step size' in the search-space. Besides, the neighborhood of each particle (individual) has to be defined.

The inspiration taken from the social-psychological sciences suggests that individuals (particles) should be influenced by their own previous experience and the experience of its neighbors. Neighborhood here refers to topological similarity in a given structure of the population. There are a number of different schemes to connect the individuals of the population. Most particle swarm implementations use one of two simple sociometric principles. The first, called *gbest* (g for global), conceptually connects all members of the population to one another. The effect of this is that each particle is influenced by the very best performance of any member of the entire population. The second, termed *lbest* (l for local), creates a neighborhood for each individual comprising itself and its k-nearest neighbors in the population.

A particle will move in a certain direction as a function of its current position $\mathbf{x}_i(t)$, its velocity $\mathbf{v}_i(t)$, the location of the particle's best success so far $\mathbf{p}_i$, and the best position found by any member of its neighborhood $\mathbf{p}_g$:

$$\mathbf{x}_i(t+1) = f\big(\mathbf{x}_i(t), \mathbf{v}_i(t+1), \mathbf{p}_i, \mathbf{p}_g\big). \tag{6}$$

The influence of the terms $\mathbf{v}_i(t)$, $\mathbf{p}_i$, and $\mathbf{p}_g$ can be summarized by a change $\mathbf{v}_i(t+1)$ to be applied at iteration $t+1$:

$$\mathbf{v}_i(t+1) = \mathbf{v}_i(t) + \boldsymbol{\varphi}_1 \otimes \big(\mathbf{p}_i - \mathbf{x}_i(t)\big) + \boldsymbol{\varphi}_2 \otimes \big(\mathbf{p}_g - \mathbf{x}_i(t)\big) \tag{7}$$

where $\boldsymbol{\varphi}_1$ and $\boldsymbol{\varphi}_2$ represent positive random vectors composed of numbers drawn from uniform distributions with a predefined upper limit: $\boldsymbol{\varphi}_1 = U(0, AC_1)$ and $\boldsymbol{\varphi}_2 = U(0, AC_2)$; $U(0, AC)$ is a vector composed of uniformly distributed random numbers, and AC is called the *acceleration constant*.

According to Kennedy [159], $AC_1 + AC_2 = 4.1$ (usually both are 2.05). The second term on the right-hand side of Eq. (7) is proportional to the difference between the particle's previous best and its current position, and the last term on the right-hand side of Eq. (7) is proportional to the difference between the neighborhood's best and the current position of the particle. The symbol $\otimes$ represents the elementwise vector multiplication.

In order to limit the change in position of a particle so that the system does not "explode", two values $\nu_{\min}$ and $\nu_{\max}$ are defined for the change $\mathbf{v}$, thus guaranteeing that the particles oscillate within some predefined boundaries: If $\nu_{\mathrm{id}} > \nu_{\max}$, then $\nu_{\mathrm{id}} = \nu_{\max}$; Else if $\nu_{\mathrm{id}} < \nu_{\min}$ then $\nu_{\mathrm{id}} = \nu_{\min}$.

Shi and Eberhart [252] suggested that by using a decreasing factor termed *inertia weight* w the PS algorithm was able to better balance exploration with exploitation:

$$\mathbf{v}_i(t+1) = w.\mathbf{v}_i(t) + \boldsymbol{\varphi}_1 \otimes \big(\mathbf{p}_i - \mathbf{x}_i(t)\big) + \boldsymbol{\varphi}_2 \otimes \big(\mathbf{p}_g - \mathbf{x}_i(t)\big) \tag{8}$$

where $\boldsymbol{\varphi}_1$ and $\boldsymbol{\varphi}_2$ were defined as previously and w was assigned an initial value but was allowed to be reduced during the iterations. The decreasing inertia weight can, for instance, be started at 0.9 and be linearly decreased until $w = 0.4$.

Another variant of the original algorithm that is currently known as the standard particle swarm algorithm [159] involves the use of a global constriction coefficient χ [50] as follows:

$$\mathbf{v}_i(t+1) = \chi\big(\mathbf{v}_i(t) + \boldsymbol{\varphi}_1 \otimes \big(\mathbf{p}_i - \mathbf{x}_i(t)\big) + \boldsymbol{\varphi}_2 \otimes \big(\mathbf{p}_g - \mathbf{x}_i(t)\big)\big) \tag{9}$$

where the parameter $\chi \approx 0.729$.

3.3.5. Scope of swarm intelligence

ACO algorithms were particularly designed to solve discrete optimization problems. Bonabeau et al. [37] claim that ACO is currently the best available heuristic for the sequential ordering problem, for real-world instances of the quadratic assignment problem, and is among the best alternatives for the vehicle and network routing problems. Good surveys of applications of ACO algorithms, including a number of main references can be found in [36,38,83–85,90].

The main problems tackled by ACO algorithms are: TSP, network routing, graph coloring, shortest common super sequence, quadratic assignment, machine scheduling, vehicle routing, multiple knapsack, frequency assignment, and sequential ordering. Particle swarm optimization algorithms have been applied to a number of numeric and parameter optimization problems, including real-world tasks. For instance, works can be found in the literature applying the PS approach to tasks like human tremor analysis, milling optimization, ingredient mix optimization, reactive power and voltage control, battery pack state-of-charge estimation, and improvised music composition [90,161].

3.3.6. Current trends and open problems

Over the last years, most works on ACO and PSO have focused on the improvement of the basic algorithms. Among the most successful variants of ACO are the 'Elitist Ant System' [81], the 'Rank-Based Ant System' [86], the 'Max–Min Ant System' [262], and the 'Hypercube Framework' [35]. Whilst in the beginning most ACO researchers were concerned with its application to engineering problems, such as the ones listed above, currently researchers in biomedics and bioinformatics have also gained interest in ACO [154,254]. Theoretical investigations into ACO include convergence analysis [128]; the comparison with other approaches, like optimal control [31] and gradient-based search [193]; and the hybridization with more classical artificial intelligence and operations research methods [34]. Dorigo and Blum [82] list a number of theoretical open problems concerning the ACO algorithm, such as their relation with other probabilistic algorithms, their convergence speed, and the development of new algorithmic components based on solid theoretical foundations. Recent research into the particle swarm optimization algorithm has emphasized its application to multi-objective, dynamic, constrained and combinatorial optimization problems [33,52,214,287]; theoretical investigations, such as scalability issues, novel operators and convergence analysis [41, 136,288]; hybridization with other approaches [230,253]. The current trends and open problems of particle swarm optimization are similar to the ones for the other techniques, like the development of more robust constraint handling techniques, automatic tuning of parameters (or guidelines for a better choice), application to dynamic, multi-objective and combinatorial problems, convergence and scalability analysis, etc. [90,159,161].

3.4. Artificial immune systems

Artificial immune systems (AIS) is a terminology that refers to adaptive systems inspired by theoretical and experimental immunology with the goal of solving problems [61,72]. They encompass any system or computational tool that extracts ideas and metaphors from the biological immune system in order to solve problems. This is also a young field of research that emerged around the mid 1980s. Its application areas range from biology to robotics. Similarly to ANNs, EAs and SI, different phenomena, processes, theories and models resulted in different types of immune algorithms, from evolutionary-like algorithms to network-like systems. This section reviews some of the pioneer works in AIS, emphasizing the three mainstreams in the field to date: negative selection algorithms, clonal selection algorithms, and immune network algorithms.

3.4.1. Biological motivation

All living beings have the ability to present resistance to disease-causing agents, known as *pathogens* (e.g., viruses, bacteria, fungi, and parasites). The primary role of the *immune system* is to protect our bodies against infections caused by pathogens [150,269]. The immune system can be divided into *innate immune system* and *adaptive immune system*, composed of diverse sets of cells, molecules and organs that work in concert to protect the organism.

The innate immune system is very important as a first line of defense against several types of pathogens and is also crucial for the regulation of the adaptive immune system. Cells belonging to the innate immune system are capable of recognizing generic molecular patterns (a type of molecular signature) that are only present in pathogens. Once a pathogen has been recognized by a cell of the innate immune system, this cell sends chemical signals to other immune cells, including those of the adaptive immune system, to start fighting against the pathogen. Therefore, the innate immune system plays a major role in providing *co-stimulatory* signals for the adaptive immune system.

Nevertheless, not all pathogens can be recognized by the innate system. Some specific pathogens are only recognized by cells and molecules of the adaptive immune system, also called specific immune system. After a certain pathogen has been eliminated by the adaptive system, the innate system plays a role in signaling the adaptive system that the foreign agent has been defeated. Once the adaptive immune system is prepared to act, it can adapt to the

invading pathogen and create specific molecular patterns to fight against the same or a similar future infection of this type. That is, the adaptive system is capable of learning (extracting information) from the infections.

The first step in promoting an *immune response*, that is, a response of the immune system against infection, is the recognition of the pathogen. Those portions of the pathogens that can promote an adaptive immune response are called *antigens*. There are several types of immune cells and molecules. Most AIS focus on the adaptive immune system with emphasis either on B-cells or T-cells, which are the most important cells from the adaptive immune system.

There are several theories used to explain how the immune system works and how it responds to pathogenic attacks. Three of the most important ones from the AIS perspective are: (1) the self-nonself discrimination theory; (2) the clonal selection theory; and (3) the immune network theory.

The self-nonself discrimination theory is based upon the principle that the immune system is capable of distinguishing between what belongs to the organism, known as self, and what does not belong to the organism, known as nonself; a principle called self/nonself discrimination [187,217]. In this case, the immune system is a recognition/action system that acts according to foreign (nonself) stimulation. Let us consider the particular case of self-nonself discrimination of T-cells in an organ called thymus. Simplistically, in the negative selection of T-cells if a self antigen (from the host) is recognized by an immature T-cell within the thymus, this cell is purged from the repertoire of T-cells, else it becomes an immunocompetent cell and is released to circulate throughout the body in the search for nonself antigens.

The clonal selection and affinity maturation theory [7,43] proposes one form of explaining how the immune system reacts to pathogens. When the organism is invaded by a pathogen, it is recognized by the receptors on some cells of the adaptive immune system. The recognizing cells are selected to proliferate at high rates and differentiate into memory cells and cells that secrete antibodies at high rates. During proliferation, the immune cells are subjected to a controlled mutation process with rates inversely proportional to the affinity with the pathogen: cells with high affinity are subject to low mutations, and vice-versa. The cumulated effect of reproduction subjected to mutation plus selection, is known as the maturation of the immune response, because it allows the immune cells to become increasingly better responding to known pathogens.

The last immunological theory to be explored here is the immune network theory [151,224]. The immune network is a network made of immune cells and molecules that interact with each other and the environment. Thus, even in the absence of pathogenic agents the immune system presents a dynamical behavior. The recognition of an immune cell by another element of the host promotes a network suppression of the recognized cell and stimulation of the recognizing cell, while the recognition of the pathogen also promotes network stimulation.

3.4.2. Design principles

The number of applications and algorithms present in the literature of AIS is vast, but some core ideas have been broadly explored, namely, *clonal selection* and *affinity maturation*, *negative selection*, and *immune networks*. Several new proposals involving mainly concepts from innate immunity and the danger theory have appeared in the last ICARIS conferences [148,203,267], but still not with a common ground among them.

To design artificial immune systems, de Castro and Timmis [72] have proposed a layered approach based on the *immune engineering* framework introduced by de Castro [67]. The immune engineering process that leads to a framework to design AIS is composed of the following basic elements [72]: (1) a *representation* for the components of the system; (2) a set of *mechanisms to evaluate the interaction* of individuals with the environment and each other. The environment is usually simulated by a set of input stimuli or patterns, one or more fitness function(s), or other means; and (3) *procedures of adaptation* that govern the dynamics and metadynamics of the system, i.e., how its behavior varies over time.

Representation The first step toward designing an AIS is to devise a scheme to create abstract models of the artificial immune cells and molecules. Any immune response requires the shape recognition of an antigen by a cell receptor. To model this shape recognition process in the immune system, A. Perelson and G. Oster [225] introduced the concept of *shape-space*. The shape-space approach assumes that all the properties of receptor molecules relevant to determining their interactions with each other and foreign antigens can be described by a *data structure*. Thus, the data structure, which usually depends upon the problem being studied, corresponds to the *generalized shape* of a molecule and is sufficient to quantify the affinity between molecules.

The most common type of data structure is an *attribute string*, which can be a real-valued vector (Euclidean shape-space), an integer string (Integer shape-space), a binary string (Hamming shape-space), or a symbolic string

(Symbolic shape-space). Assume the case in which any receptor molecule of an immune cell is called an antibody and is represented by the set of coordinates $\mathbf{Ab} = (Ab_1, Ab_2, \ldots, Ab_L)$; an antigen is given by $\mathbf{Ag} = (Ag_1, Ag_2, \ldots, Ag_L)$, where boldface letters correspond to attribute strings. The interaction of antibodies, or of an antibody and an antigen, can be evaluated via a *distance measure*, also termed *affinity measure*, between their corresponding attribute strings.

Evaluating interactions For real-valued shape-spaces, for instance, the most common affinity measures are the Euclidean (E) and Manhattan (M) distances:

$$E = \sqrt{\sum_{i=1}^{L} (Ab_i - Ag_i)^2}, \tag{10}$$

$$M = \sum_{i=1}^{L} |Ab_i - Ag_i|. \tag{11}$$

For Hamming shape-spaces, the Hamming distance can be used to evaluate the affinity between two cells:

$$D = \sum_{i=1}^{L} \delta_i, \quad \text{where } \delta_i = \begin{cases} 1 & \text{if } Ab_i \neq Ag_i, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

Two other affinity measures are broadly used in Hamming shape-spaces, namely, the so-called r-contiguous bits (rcb) and the r-chunks matching rules. In the rcb case, the number of contiguous matching bits determines the affinity between two strings. The match between two strings is performed and the affinity corresponds to the size of the longer sequence of matching (equal) bits. In r-chunks, only r contiguous positions are specified instead of specifying all L attributes of a string. An r-chunk detector can be thought of as a string of r bits together with its starting position within the string, known as its window [91].

Procedures of adaptation Given a suitable representation for the immune cells and molecules, and how to evaluate their interactions, it is possible to identify some general-purpose immune algorithms. These algorithms can be separated into two classes: population-based and network-based. The first class involves all algorithms that do not take into account the immune network, and the network-based algorithms are all those inspired by the network theory of the immune system. Five main types of algorithms are available in the literature:

- *Bone marrow*: used to generate populations of immune cells and molecules to be used in AIS and possibly in other approaches, such as genetic algorithms (e.g., [137,212,226]).
- *Negative selection*: used to define a set of detectors (e.g., attribute strings) to perform, mainly, anomaly detection (e.g., [104,120,141]).
- *Clonal selection*: used to generate repertoires of immune cells driven by antigens. It regulates the expansion, genetic variation, and selection of attribute strings [59,75,105,157].
- *Continuous immune network models*: used to simulate dynamic immune networks in continuous environments (e.g., [96,272]).
- *Discrete immune network models*: used to simulate dynamic immune networks in discrete environments (e.g., [74,112,201,268]).

3.4.3. Scope of artificial immune systems

One of the most remarkable features of the field of artificial immune systems is its broad applicability. There are AIS applied to domains such as machine-learning and pattern recognition, anomaly detection and the security of information systems, data analysis (knowledge discovery in databases, clustering, etc.), agent-based systems, scheduling, autonomous navigation and control, search and optimization, and artificial life. There are a few survey papers, reports, and book chapters that can be used by the reader to find information on where and when AIS have been developed and applied. Chapters 4 and 6 of the book by L.N. de Castro and J. Timmis [72] survey a vast amount of works on AIS. Y. Ishida [147], D. Dasgupta and Attoh-Okine [62], L.N. de Castro and F. Von Zuben [73], Dasgupta et al. [64], L.N. de Castro [68], and U. Aickelin et al. [6] all have published survey papers, reports or bibliographies of artificial immune systems and their applications.

3.4.4. Current trends and open problems

Together with swarm intelligence, artificial immune systems are one of the most recent computing inspired by nature approaches. Over the last few years, important investigations in the field have focused on the proposal of theoretical frameworks for the design of AIS [72,260]; theoretical investigations into existing AIS [108,114,274]; danger theory [3,5,248]; new models of immunological phenomena [124,198]; and the hybrids with other approaches (e.g., [166,199]). Following the avenues pursued by current research, the main open problems and challenges include to strengthen theoretical developments, to improve existing algorithms and analyze their behavior and performance, to take a deeper look into immunology in the search for a more accurate biological plausibility and novel ideas [4,71, 114,132].

4. The synthesis of natural phenomena in computers

While nature-inspired computing is basically aimed at solving complex problems, the second branch of natural computing provides new tools for the synthesis and study of natural phenomena that can be used to test biological theories that cannot be tested via the traditional experimental and analytic techniques. It is in most cases a synthetic approach aimed at simulating in computers natural phenomena or known patterns and behaviors. There is also a complementary relationship between biological theory and the synthetic processes of the simulation and emulation of nature in computers. Theoretical studies suggest how the synthesis can be achieved, while the application of the theory in the synthesis may be a test for the theory itself. There are basically two main approaches to the simulation and emulation of nature in computers: by using tools for studying the *fractal geometry of nature* or by using *artificial life* techniques.

4.1. Fractal geometry

One major breakthrough in the modeling and synthesis of natural patterns and structures is the recognition that nature is fractal [93,99,177,183,223]. In general, fractals are characterized by infinite details, infinite length, self-similarity, *fractal dimensions*, and the absence of smoothness or derivative. Nature provides many examples of fractals, for instance, ferns, coastlines, mountains, cauliflowers, broccoli, our lungs, our circulatory system, our brains, and our kidneys. This section starts by introducing the concept of fractal dimension and follows with the presentation of two techniques for modeling natural phenomena: *cellular automata* [146,280] and *L-systems* [178].

4.1.1. Fractal dimension

B. Mandelbrot [183] coined the term *fractal* to identify a family of shapes that describe irregular and fragmented patterns in nature, thus differentiating them from the pure geometric forms from the Euclidean geometry. The word fractal comes from the Latin adjective *fractus*, whose corresponding verb *frangere* means 'to break' or to create irregular fragments.

Self-similarity is a core concept in fractal geometry. Consider an example of a natural structure that presents this property: a broccoli. By removing some branches of the broccoli and comparing them with the whole broccoli, one can note that the branches are very much like the whole, but in a smaller scale. These branches can be further decomposed into smaller branches, which are again very similar to the whole broccoli. This self-similarity carries through for a number of dissections. In a mathematical idealization, the property of self-similarity of a fractal may be continued through infinitely many stages.

We learn early in the school days that points are zero-dimensional, lines and curves are one-dimensional, planes and surfaces are two-dimensional, and solids, such as cubes and spheres, are three-dimensional. Roughly speaking, a set is d-dimensional if d independent variables (coordinates) are needed to describe the neighborhood of any point. This notion of dimension is called the *topological dimension* of a set. The discovery of some 'mathematical monsters', such as space-filling curves, had a major impact in the development of the concept of dimension. These structures questioned the intuitive perception of curves as one-dimensional objects because they filled the plane, and thus they are intuitively perceived as bi-dimensional objects.

If the dimension of the object is known, and this is the case for the Euclidean shapes, powers or exponents allow us to work out how many smaller copies of itself the object contains, of any particular size. In the case of regular shapes, such as lines, squares, and cubes, a d-dimensional shape is composed of $N \times (1/m)$ copies of itself, where each copy

has a size m in relation to the original shape; m is called the *reduction factor*. The relationship is, thus, given by $N = (1/m)^d$. This idea of the relation between the logarithm of the number of copies of an object and the logarithm of the size of its copies suggested a generalization of the concept of dimension, which allows fractional values. This dimension is known as the *self-similarity dimension*.

$$N = (1/m)^d \implies d = \frac{\log N}{\log 1/m}. \tag{13}$$

Mandelbrot noticed that the slope of the resultant line plotted by the logarithm of the length of a measuring stick against the total length is but the *Hausdorff dimension*. In 1919, F. Hausdorff extended the notion of similarity dimension, given by Eq. (13), to cover all shapes, not just strictly self-similar shapes. This *fractal dimension* describes the fractal complexity of an object.

4.1.2. Cellular automata

Cellular automata (CA) are a class of spatially and temporally discrete, deterministic mathematical systems characterized by local interaction and an inherently parallel form of evolution in time [146]. For short, a cellular automaton can be defined as a dynamical system that is discrete in both space and time. The formalism for cellular automata was introduced by John von Neumann (with some suggestions from Stanislaw Ulam) in the late 1940s and early 1950s as a framework to study self-reproduction.

A d-dimensional cellular automaton consists of a d-dimensional grid of cells, each of which can take on a value from a finite, usually small, set of integers. The value of each cell at time step $t + 1$ is a function of the values of a small local neighborhood of cells at time t. The cells update their state simultaneously according to a given local rule [186].

Formally, a cellular automaton C can be defined as a 5-tuple

$$C = \langle S, \mathbf{s}_0, G, d, f \rangle \tag{14}$$

where S is a finite set of states, $\mathbf{s}_0 \in S$ are the initial states of the CA, G is the cellular neighborhood, $d \in Z^+$ is the dimension of C, and f is the local cellular interaction rule, also referred to as the *transition function*.

Given the position of a cell $\mathbf{i}$ in a regular d-dimensional uniform *lattice*, or *grid*, its neighborhood G is defined by

$$G_{\mathbf{i}} = \{\mathbf{i}, \mathbf{i} + \mathbf{r}_1, \mathbf{i} + \mathbf{r}_2, \ldots, \mathbf{i} + \mathbf{r}_n\} \tag{15}$$

where n is a fixed parameter that determines the neighborhood size, and $\mathbf{r}_j$ is a fixed vector in the d-dimensional space. The local transition rule f

$$f : S^n \to S \tag{16}$$

maps the state $s_{\mathbf{i}} \in S$ of a given cell $\mathbf{i}$ into another state from the set S, as a function of the states of the cells in the neighborhood $G_{\mathbf{i}}$. In a uniform CA, f is identical for all cells, whereas in nonuniform CA, f may differ from one cell to another, i.e., f depends on $\mathbf{i}$, $f_{\mathbf{i}}$.

For a finite-size CA of size N, where N is the number of cells in the CA, a configuration of the grid at time t is defined as

$$C(t) = \big(s_0(t), s_1(t), \ldots, s_{N-1}(t)\big) \tag{17}$$

where $s_{\mathbf{i}}(t)$ is the state of cell $\mathbf{i}$ at time t. The progression of the CA in time is then given by the iteration of the global mapping F

$$F : C(t) \to C(t + 1), \quad t = 0, 1, \ldots . \tag{18}$$

Through the simultaneous application in each cell of the local transition rule f, the global dynamics of the CA can be described as a directed graph, referred to as the CA's *state space*.

4.1.3. Lindenmayer systems

Development at the multicellular level consists of the generation of structures by cell division, cell enlargement, cell differentiation, and cell death taking place at determined times and places in the entire life of the organism. A. Lindenmayer [178] introduced a formalism to simulate the development of multicellular organisms, later referred to as Lindenmayer systems or simply *L-systems*. The development of the mathematical theory of L-systems was followed by its application to the modeling of plants, which is the main focus of this section.

A *string* or *word OL-system* is defined as the ordered triplet $G = \langle V, \omega, P \rangle$, where V is the *alphabet* of the system, $\omega \in V^+$ is a nonempty word called the *axiom*, and $P \subset V \times V^*$ is a finite set of productions [231]. A production $(a, \chi) \in P$ is written as $a \to \chi$. The letter a and the word χ are called the *predecessor* and the *successor* of this production, respectively. It is assumed that for any letter of the alphabet there is at least one word from the set V^* such that $a \to \chi$; that is, $\exists \chi \in V^* \mid a \to \chi, \forall a \in V$. An OL-system is *deterministic*, noted *DOL-system*, if and only if for each $a \in V$ there is exactly one $\chi \in V^*$ such that $a \to \chi$.

Let $\mu = a_1 \dots a_m$ be an arbitrary word over V. The word $\nu = \chi_1 \dots \chi_m \in V^*$ is *directly derived* from (or *generated* by) μ, noted $\mu \Rightarrow \nu$, if and only if $a_i \to \chi_i$ for all $i = 1, \dots, m$. A word ν is generated by G in a derivation of *length* n if there exists a *developmental sequence* of words $\mu_0, \mu_1, \dots, \mu_n$ such that $\mu_0 = \omega$, $\mu_n = \nu$ and $\mu_0 \Rightarrow \mu_1 \Rightarrow \cdots \Rightarrow \mu_n$.

In the description presented above, L-systems were used to generate a sequence of words or strings. The *geometric interpretation* of these words can be used to generate schematic images of diverse natural patterns. A modified *turtle graphics* [215] language can be used to interpret the words generated by L-systems [266]. In turtle graphics plotting is performed by a smart little turtle that follows certain commands.

The basic idea of turtle interpretation is as follows. The *state* of the *turtle* is defined as a triplet (x, y, α), where the Cartesian coordinates (x, y) represent the turtle's position, and the angle α, called the *heading*, is interpreted as the direction to which the turtle is facing. Given the *step size* d and the *angle increment* δ, the turtle can respond to commands represented by the following symbols:

F, G	Move forward a step of length d. The state of the turtle changes to (x', y', α), where $x' = x + d.\cos\alpha$, and $y' = y + d.\sin\alpha$. A line segment between points (x, y) and (x', y') is drawn.
f, g	Move forward a step of length d without drawing a line.
$+$	Turn right by angle δ. The next state of the turtle is $(x, y, \alpha + \delta)$. The positive orientation of angles is clockwise.
$-$	Turn left by angle δ. The next state of the turtle is $(x, y, \alpha - \delta)$. The negative orientation of angles is counterclockwise.

In his 1968 papers, Lindenmayer [178] introduced a notation for representing graph-theoretic trees using strings with *brackets*. The *bracketed L-systems* extend an L-system alphabet by the set {[,]}. The motivation was to formally describe branching structures found in many plants, from algae to trees, using the general framework of L-systems. Subsequently, the geometric interpretation of bracketed L-systems was used with the purpose of presenting modeled structures in the form of computer-generated figures.

The two new symbols '[' and ']' are interpreted by the turtle as follows:

[Save the current state (x, y, α) of the turtle for later use onto a stack of saved states.
] Remove the last saved state from the stack and use it to restore the turtle's last state. No line is drawn, although in general the position of the turtle changes.

4.1.4. Scope of the cellular automata and L-systems

Cellular automata are used in biology, chemistry, physics, in the design of parallel computers and image processing, as well as image generation. CA also became attractive to people doing research in artificial life, artificial intelligence, and theoretical biology, and have proven to be useful idealizations of the dynamical behavior of many real complex systems, including physical fluids, neural networks, plant growth, molecular dynamical systems, natural ecologies, immune system modeling, military command and control networks, forest fires, economy, among others. Because of their underlying simplicity, CA are also powerful conceptual tools with which to study general pattern formation. They have already provided critical insights into the self-organized patterns of chemical reaction-diffusion systems, crystal growth, seashells, and phase-transition-like phenomena in vehicular traffic flow, to name but a few examples. There are a number of applications in which L-systems play important roles as models of natural patterns [241]: reconstruction of extinct plant species, identification of plant responses to pest attacks, structural models of trees integrated in

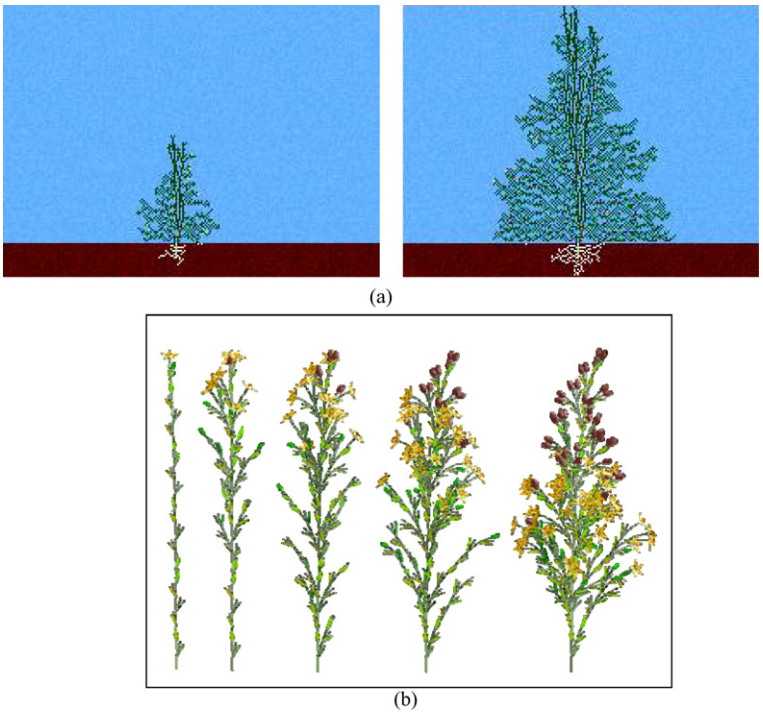

Fig. 5. Examples of patterns generated by the fractal geometry methods presented. (a) Two stages of the growth of a tree generated by a cellular automaton. Note that the tree and its roots are both growing. Generated with CAFUN, freely available for download from the site: http://www.cafun.de. (b) Three-dimensional rendering of a developmental sequence of the plant *Mycelis muralis* simulated using L-system (reproduced from [231] with permission).

more complex forest ecosystem simulations, design of new varieties of plants, synthesis of seashells and other natural patterns, modeling of developmental processes, classification of branching patterns in inflorescences, crop yield prediction, modeling of fractal patterns, inflorescence description, simulation of fungal growth, and computer aided learning for farm managers. Also, they are present in other non-biological fields like formal language theory, tumor growth, and musical composition. Fig. 5 illustrates some patterns generated with CA and L-systems.

4.1.5. Current trends and open problems

This paper emphasized the use of fractal geometry to synthesize plants and natural sceneries. However, fractals have several other approaches and applications, among which the main current trends are the use of fractals as function approximators [19,46,200], their applications in economics and finance [163,184,185,189], several technological applications, such as antenna design, data mining and image processing [18,53,179,257,279,282], applications in biology and medical sciences [16,58,130,133,145,290], applications in forecasting, such as weather forecasting [172, 202], and applications in arts and games [164,251]. The open problems include theoretical investigations into fractals and determining the fractal dimension of natural patterns.

4.2. Artificial life

Artificial life (ALife) performs one step toward the challenging achievement of synthesizing life-like phenomena and, ultimately, life. As put by C. Langton in the first of a series of workshops specifically devoted to the study of ALife:

"Artificial Life is the study of man-made systems that exhibit behaviors characteristic of natural living systems. It complements the traditional biological sciences concerned with the *analysis* of living organisms by attempting to *synthesize* life-like behaviors within computers and other artificial media. By extending the empirical foundation upon which biology is based *beyond* the carbon-chain life that has evolved on Earth, Artificial Life can contribute to theoretical biology by locating *life-as-we-know-it* within the larger picture of *life-as-it-could-be*" [176, p. 1, original emphasis].

Some important ideas lye behind the study of artificial life:

- Life is viewed as a dynamic process with certain universal features that are not dependent upon the matter. Thus, life is an emergent property of the organization of matter, rather than a property of the matter itself.
- ALife employs a synthetic approach to the study and creation of life(-as-it-could-be).

4.2.1. The essence of life

One central question raised in artificial life regards how can we claim that a given system is a realization, emulation, or simulation of life. While simulations are assessed by how well they generate similar morphologies, patterns or behaviors of some specified aspects of a system, a realization is a literal, substantive and functional device that does not explicitly involve the specification of the global structure or behavior of the system [219]. The concept of emergence—properties of the whole that are not possessed by nor directly derivable from any of the parts—is fundamental in ALife research [89,152].

What is life is a question that has been puzzling us for centuries. Definitions vary from philosophy to biology to psychology, and the literature on the subject is also vast (cf. [88,188,237,246]). Despite some attempts to define life, some ALife approaches involve assembling a list of properties of life, and then testing candidates on the basis of whether or not they exhibit any item of the list [95,232]. Another approach is to provide relationships between an artificial and a natural agent [156]. The question is how can we identify something as life-as-it-could-be (ALife) rather than something exceedingly interesting but not alive?

Farmer and Belin [95] provided an eight-item list (argued to be imprecise and incomplete) of properties associated with life:

1. *Life is a pattern in space-time*, rather than a specific material or object.
2. *Life involves self-reproduction*. If not in the organism itself, at least some related organism must be capable of reproduction (mules cannot reproduce but none would argue they are not alive).
3. *Information storage of a self-representation*. For instance, in DNA molecules.
4. *Metabolism*, responsible for converting matter and energy from the environment into the pattern and activities of the organism.
5. *Functional interactions with the environment*. Living organisms can respond to or anticipate changes in the environment.
6. *Interdependence of parts*. The components of the living system depend on one another.
7. *Stability under perturbation* and insensitivity to small changes.
8. *The ability to evolve*, which is not a property of a single individual, but of a population of individuals.

Despite whether we consider a system as alive because it exhibits any or all of these properties of life, or because it relates to a natural organism (life-as-we-know-it), what is more important is the recognition that it is possible to create simple disembodied realizations or instances of (specific properties of) life in computers and other artificial media.

4.2.2. Examples of ALife projects

This section contains two examples of artificial life systems: the (1) *StarLogo* programming language; and (2) *Boids*.

Turtles, termites, and traffic jams M. Resnick [238] developed a computational tool, named *StarLogo*, that people can use to construct, play with, and think about self-organized systems. Inspired by *cellular automata*, StarLogo encourages constructivist thinking in which the actions of individual agents could arise from interactions of individual agents. This would result in the possibility of modeling many types of objects in the world, such as an ant in a colony, a car in a traffic jam, an antibody in the immune system, and a molecule in a gas.

StarLogo was thus designed as a massively parallel language with an environment capable of accommodating a great number of agents, named *turtles*, simultaneously. Many senses can be assigned to each turtle; for instance, a turtle can perceive and act on its surrounding environment. The environment in which the turtles are placed, termed *world*, is active, and is divided into square sections called *patches*. It means that patches can store information about

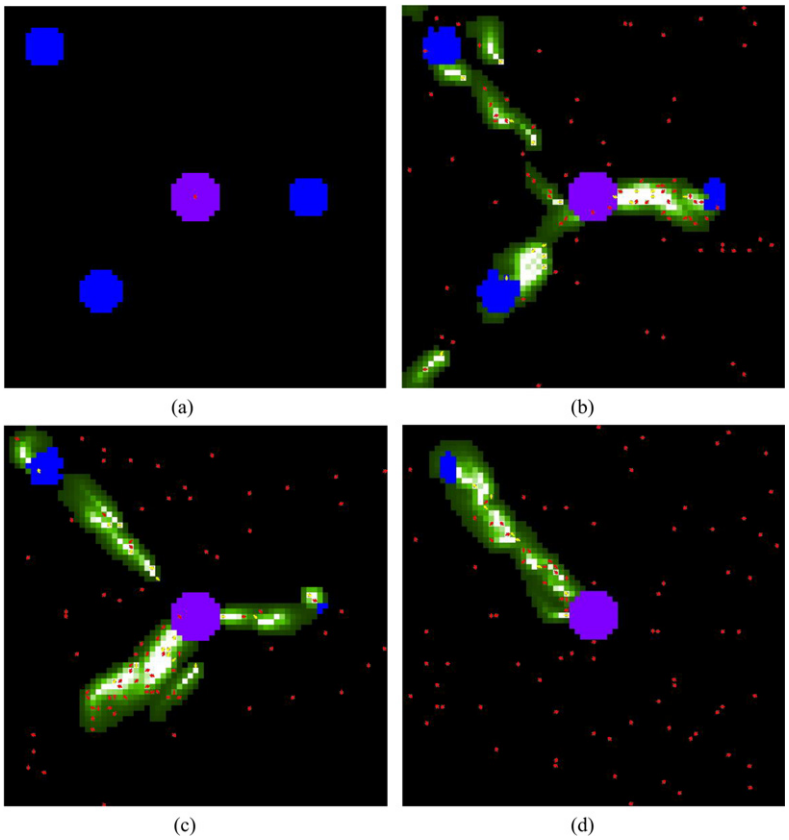

Fig. 6. Trail-laying trail following behavior of ants. (a) Initial environmental setup: nest in the center and food sources spread around it with varying distances. (b) When the ants find a food source they move back to the nest carrying food and leaving a trail of pheromone (light patches). (c) The ants explore the environment, exploiting first and more strongly those food sources closer to the nest. (d) The closer food sources are depleted before the farther one. Generated with StarLogo, freely available for download from the site: http://education.mit.edu/starlogo/.

the activities of the turtles and execute StarLogo instructions just as turtles do. Thus the environment is given equal status as being alive as the agents that inhabit it.

Several projects inspired by real-world or natural systems, such as ant colonies, traffic jams, and termite mound building, are presented in [238], including the computer codes necessary to run them. These demo projects are also available with the software. An interesting example is the *Ants* project that explores an ant colony foraging for food. As discussed in Section 3.3.1, the ants show a trail-laying trail-following behavior that enables them to find the shortest food source in relation to the nest. Fig. 6(a) depicts the standard setup of the *Ants* project, with a nest in the center of the picture and three food sources spread around the nest with varying distances. It can be observed that after some iterations, Fig. 6(b), the ants find the food sources and start exploring them proportionally to their distance to the nest; closer sources are explored first and more strongly, while farther sources are explored later and more weakly (Fig. 6(c)). Only when the closer food sources are depleted the farther one is fully exploited (Fig. 6(d)).

Flocks, herds, and schools The motions of a flock of birds, a herd of land animals, and a school of fish are very beautiful social behaviors intriguing to contemplate. Producing an animated computer graphic portrayal of these social behaviors is not an easy task, mainly if one tries to script the path of a large number of agents using the more traditional computer animation techniques; that is, by carefully describing the behavior of each bird, animal, or fish. In a classic ALife work, Reynolds [239] demonstrated that the apparently intentional, centralized behavior of a flock of birds can be described by a small set of rules governing only the behavior of individual agents, acting solely on the basis of their own local perception of the environment. The overall resultant behavior is an emergent property of a set of interacting elements following simple local rules.

To create a flocking behavior, a virtual agent called *boid* must be aware of itself and its nearest neighbors, suggesting that it can flock with any number of flock mates. The basic flocking model consists of three simple steering behaviors [240], which can be stated as rules describing how an individual boid maneuvers based on the positions and velocities of its nearby flock mates: (1) *collision avoidance and separation*: attempt to avoid collisions with nearby flock mates; (2) *velocity matching and alignment*: attempt to match velocity with nearby flock mates, and to move towards the average heading of nearby flock mates; and (3) *flock centering or cohesion*: attempt to stay close to the average position of nearby flock mates.

The flocking model described gives boids a very close pattern of behavior to that observed in real flocks of birds. Boids released near one another start flocking together, cavorting, and jostling for position. They stay close to one another, but always maintain prudent distance from the neighbors, and the flock quickly becomes biased with its members heading in approximately the same direction at approximately the same speed. When they change direction, they do it in coordination. Solitary boids and smaller groups tend to cluster together in larger flocks and, in the presence of obstacles, larger flocks can (temporarily) split into smaller flocks.

4.2.3. Scope of artificial life

Artificial life is of interest to biologists mainly because its models can shed light on biological phenomena. It is relevant to engineers because it offers methods to generate and control complex behaviors that are difficult to generate or regulate using traditional techniques. It is important to people in general because it allows the creation of novel technologies that can be used as entertainment, scientific tools, and to assist humans in the accomplishment of difficult or unpleasant tasks. Artificial life has many other faces involving various aspects of cognitive science, economics, art, physics, chemistry, and even ethics.

4.2.4. Current trends and open problems

The inspiration in *life-as-we-know-it* results in benefits for science and our daily life as well. Current ALife research includes its applications in arts and games [98,182,190], investigations into the evolutionary nature of life as well as into the realization of life [182,207,210,233,237,278], studies concerning the evolution of language [48,165], and applications in technological areas, such as robotics [45,171,206]. Focusing the more fundamental and theoretical aspects of artificial life and not taking into account the technological challenges of ALife, Bedau et al. [22] point out the following challenges for the field: to study and understand how life emerges from the inanimate, what are the limitations and potentials of living beings, and how is life related to mind, machines and culture.

5. Computing with new natural materials

Computing with new natural materials is concerned with new computing methods based on other natural material than silicon. The history of computer technology has involved a sequence of changes from one type of realization to another; from gears to relays to valves to transistors to integrated circuits. Nowadays, a single silicon chip can contain millions of logic gates. This miniaturization of the most basic information processing elements is inevitably going to reach a state where logic gates will be so small so as to be made of atoms. Computing with natural materials is the approach that promises to bring a major change in the current computing technology. Motivated by the need to identify alternative media for computing, researchers are now trying to design new computers based on molecules, such as DNA and RNA, or quantum theory. These ideas resulted in what is now known as *molecular computing* [44, 123,220,221,256] and *quantum computing* or *quantum computation* [138,204,229].

5.1. DNA computing

The ‘machine’ language that describes the objects and processes of living systems contains four letters $\{A, C, T, G\}$, and the ‘text’ that describes a person has a great number of characters. These form the basis of the DNA molecules that contain the genetic information of all living beings. DNA is also the main component of *DNA computing*.

DNA computing is one particular component of a large field called *molecular computing* that can be broadly defined as the use of (bio)molecules and biomolecular operations to solve problems and to perform computation. It was introduced by L. Adleman in 1994 when he solved an NP-complete problem using DNA molecules and biomolecular

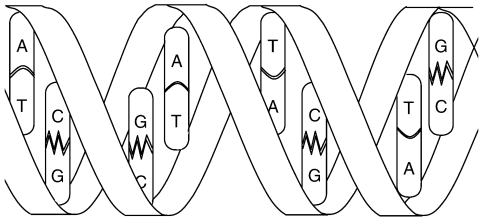

Fig. 7. The double helix of DNA. A is complementary to T, and C is complementary to G.

techniques for manipulating DNA. The basic idea is that it is possible to apply operations to a set of (bio)molecules, resulting in interesting and practical performances.

It is possible to divide the DNA computing models in two major classes [291]. A first class, commonly referred to as *filtering models* [8], which includes models based on operations that are successfully implemented in the laboratory. A second class composed of the so-called *formal models*, such as the *splicing systems* [135] and the *sticker systems* [243], whose properties are much easier to study, but only the first steps have been taken toward their practical implementation.

5.1.1. Biological motivation

The genetic material is contained in the cell nucleus and is complexed with proteins and organized into linear structures called *chromosomes*. Chromosomes are composed of genes, which, by themselves, are segments of a helix molecule called *deoxyribonucleic acid*, or DNA for short. All the genetic information in cellular organisms is stored in DNA, which consists of *polymer chains*, commonly referred to as *DNA strands*, of four simple nucleic acid units, called *deoxyribonucleotides* or simply *nucleotides*. There are four nucleotides found in DNA. Each nucleotide consists of three parts: one *base molecule*, a *sugar* (deoxyribose in DNA), and a *phosphate group*. The four bases are *adenine* (A), *guanine* (G), *cytosine* (C), and *thymine* (T). As the nucleotides differ only by their bases, they are often called bases (Fig. 7).

Nucleotides can link together in two different ways [221]:

- The phosphate group of one nucleotide is joined with the hydroxyl group of the other forming a *covalent bond*.
- The base of one nucleotide interacts with the base of the other to form a *hydrogen bond*, which is a bond weaker than the covalent bond.

The bonds between the bases can only occur by the pairwise attraction of the following bases: A binds with T, and G binds with C. This is called the *Watson–Crick complementarity*, after J.D. Watson and F.H.C. Crick who discovered the double helix structure of DNA.

All DNA computing techniques apply a specific set of biological operations to a set of strands. These operations are all commonly used by molecular biologists, and the most important of them, in the context of DNA computing, are: (1) *Denaturation*: separates DNA strands; (2) *Annealing*: fuses DNA strands; (3) *Polymerase extension*: fills in incomplete strands; (4) *Nuclease degradation*: shortens DNA molecules; (5) *Endonucleases*: cut DNA molecules; (6) *Ligation*: links DNA molecules; (7) *Modifying nucleotides*: inserts or deletes short subsequences; (8) *Amplification*: multiplies DNA molecules. Usually, a technique called *polymerase chain reaction* (PCR) is used to make multiple copies of a subset of the strands present; (9) *Gel electrophoresis*: measures the length of DNA molecules and separates them by length; (10) *Filtering*: separates or extracts specific molecules; (11) *Synthesis*: creates DNA molecules; and (12) *Sequencing*: reads out the sequence of a DNA molecule.

5.1.2. A filtering model

In all filtering models, a computation consists of a sequence of operations on finite multi-sets of strings that usually starts and ends with a single multi-set. By initializing a multi-set and applying specific operations to it, new or modified multi-sets are generated. The computation then proceeds by *filtering out* strings that cannot be a solution [8].

Speculations about the possibility of using DNA molecules to perform computation date back to the early 1970s and maybe even earlier (cf. [26,54]). However, none of these insights was followed by practical attempts at real world implementations. The first successful experiment involving the use of DNA molecules and DNA manipulation

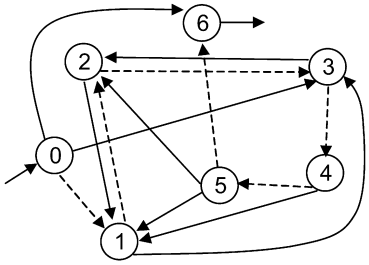

Fig. 8. Instance of the Hamiltonian path problem solved by Adleman's pioneering experiment. The dashed arrows correspond to the Hamiltonian path of this instance.

techniques for computing was reported by L.M. Adleman in 1994 [1]. In that paper, Adleman solved a small instance of the Hamiltonian path problem (HPP) in a directed graph using purely biochemical means.

The Hamiltonian path problem can be explained as follows. A directed graph G with designated vertices v_{in} and v_{out}, is said to have a Hamiltonian path if and only if there is a sequence of compatible directed edges $e_1, e_2, \ldots, e_z$ (i.e., a path) that begins at v_{in}, ends at v_{out}, and passes through each vertex exactly once. Fig. 8 illustrates the instance of the Hamiltonian path problem used by Adleman [1] in his pioneering implementation of DNA computing. In this graph, $v_{\text{in}} = 0$ and $v_{\text{out}} = 6$, and a Hamiltonian path is given by the following sequence of edges: $0 \to 1 \to 2 \to 3 \to 4 \to 5 \to 6$. This can be easily verified by inspection.

To solve this problem Adleman used the following deterministic algorithm:

Step 1: generate random paths through the graph.
Step 2: keep only those paths that begin with v_{in} and end with v_{out}.
Step 3: if the graph has n vertices, then keep only those paths that enter exactly n vertices.
Step 4: keep only those paths that enter all the vertices of the graph at least once.
Step 5: if any path remains, say YES; else, say NO.

Adleman translated this algorithm, step by step, into molecular biology. Before generating the random paths through the graph, it is necessary to decide how these are going to be encoded using DNA molecules. Adleman chose to encode each vertex of the graph by a single stranded sequence of nucleotides of length 20 (a 20 mer). The codes were constructed at random, and the length 20 was chosen so as to ensure different codes. A large number of these oligonucleotides were generated by PCR and placed in a test tube. The edges are encoded as follows: if there is an edge from vertex i to vertex j, and the codes of these vertices are $v_i = a_i b_i$ and $v_j = a_j b_j$, where a_i, b_i, a_j, b_j are sequences of length 10, then the edge $i \to j$ is encoded by the Watson–Crick complement of the sequence $b_i a_j$.

In order to link the edges to form paths, oligonucleotides $\bar{O}_i$ complementary to those representing the edges (O_i) had to be synthesized. An enzymatic ligation reaction was then carried out, linking the 20 mer strands encoding the edges so as to form random paths through the graph, corresponding to Step 1 of the algorithm. To implement Step 2, the product of Step 1 was amplified by polymerase chain reaction (PCR) using as primers $\bar{O}_0$ and $\bar{O}_6$. Therefore, only those molecules encoding paths that begin with vertex 0 and end with vertex 6 were amplified. A filtering operation can then be used to separate the strands that start with vertex 0 and end with vertex 6. Then, the DNA molecules were separated according to their length by gel electrophoresis. The band on the gel, which by comparison with a molecular weight marker was identified as consisting of strands with 140 bp (7 vertices) was separated from the gel and the DNA was extracted. Repeated cycles of PCR and electrophoresis were used to purify the product further. At the end of this step, there is a set of molecules that start with 0, end with 6, and pass through 7 vertices. To implement Step 4 of the algorithm, single stranded DNA was probed with complementary oligonucleotides attached to magnetic beads. Finally, to obtain the YES/NO answer of Step 5, one still has to amplify the product of Step 4 by PCR and analyze it by gel electrophoresis.

5.1.3. Scope of DNA computing

DNA computing was originally proposed as a problem solving technique. The massive parallelism and miniaturization of DNA suggest a number of problems that can be solved by DNA computing, in particular NP-complete

problems. Since any instance of any NP-complete problem can be expressed in terms of an instance of any NP-complete problem, the DNA computing solution to the Hamiltonian path problem described in this paper implicitly provides sufficient computational power to solve any problem in NP [113]. Based on these ideas, DNA computing has been used to solve search problems like graph coloring, shortest common superstring, integer factorization, protein conformation, maximum clique, and many others. Works can also be found in the literature proposing the realization of associative memories using DNA [20]; the solution of cryptographic problems [39]; the development of DNA-based algorithms for addition and matrix multiplication [126,211]; parallel machines [235]; and the DNA implementation of, or hybridization with, nature-inspired computing approaches [77,181,195,197].

5.1.4. Current trends and open problems

DNA computing is just of one the branches of a larger field called molecular computing, which involves the use of biomolecules to perform computation. The main current trends in molecular computing, more specifically in DNA computing, include theoretical investigations, such as the demonstration of the universality of the computations performed by DNA computers [263] and the study of formal models of DNA computing [222,291]; applications, like in cryptography [2,116,127], the solution of complex problems in computer science [23,213]; the design of mechanical devices [234,264]; and the design and implementation of real DNA computers [9,11]. All these involve the proposal of methods to improve the efficiency of DNA-based computing, such as error correction mechanisms, the design of efficient encoding schemes, studies into the scalability and how to explore the inherent parallelism of DNA computation, the discovery of new problems to which DNA computing can be applied, the investigation into the use of other molecules than DNA to perform computation, and the complexity analysis of DNA computations [10,92,129,181,236,244].

5.2. Quantum computing

The attempts to describe atomic events using classical mechanics led to contradictions, for atoms and molecules at very small distances do not behave following classical mechanics. Quantum physics is the theory that explains the behavior of objects with atomic scales. *Quantum mechanics* is a terminology used to refer to the mathematical framework that describes quantum physics [138,204,229]. It incorporates new principles into *classical physics*, such as *energy quantization*, *duality*, *uncertainty*, and *probability*.

5.2.1. Quantum mechanics

A common formulation of quantum mechanics is the transformation theory proposed by P. Dirac [79], in which the instantaneous state of a quantum system is described by a quantum state that encodes the probabilities associated with all *observables*, i.e., measurable properties.

The standard notation used to represent a vector in a vector space is as follows:

$$|\mathbf{x}\rangle \tag{19}$$

where $\mathbf{x}$ is a label for the vector. The notation $|\cdot\rangle$, usually referred to as *ket*, indicates that $\mathbf{x}$ is a (column) vector. Every ket $|\mathbf{x}\rangle$ has a dual *bra* $\langle\mathbf{x}|$, which is the conjugate transpose of $|\mathbf{x}\rangle$: $\langle\mathbf{x}| = |\mathbf{x}\rangle^{\dagger}$.

The state of a quantum system is described by a unit-length vector in a Hilbert space, i.e., an n-dimensional complex vector space H^n corresponding to the *state space* of the system. The state space of a quantum system, known as a *quantum state space*, can be described in terms of vector and matrices or using a standard notation called *bra-ket* (*bracket*) *notation* or *Dirac notation* due to its proponent, P. Dirac [79]. The terminology 'bracket notation' is so-called because the *inner product* of two states is represented by a *bracket*, $\langle\mathbf{y}|\mathbf{x}\rangle$, which consists of a left part $\langle\mathbf{y}|$, the *bra*, and a right part $|\mathbf{x}\rangle$, the *ket*. The combination of $\langle\mathbf{y}|$ and $|\mathbf{x}\rangle$, $\langle\mathbf{y}||\mathbf{x}\rangle$ also represented as $\langle\mathbf{y}|\mathbf{x}\rangle$, corresponds to the inner product between the two vectors, and results in a complex number. The notation $|\mathbf{x}\rangle\langle\mathbf{y}|$ represents the *outer product* of vectors $|\mathbf{x}\rangle$ and $\langle\mathbf{y}|$, and results in a matrix.

Any state of a quantum system can be written as a linear combination of a number of *basis states*:

$$\mathbf{c}_1|\mathbf{x}_1\rangle + \mathbf{c}_2|\mathbf{x}_2\rangle + \cdots + \mathbf{c}_n|\mathbf{x}_n\rangle \tag{20}$$

where $\mathbf{c}_i$, $i = 1, \ldots, n$, are complex numbers called *amplitudes*, and $\sum_i |\mathbf{c}_i|^2 = 1$. Each value $|\mathbf{c}_i|^2$ corresponds to the probability that the system is found at state $\mathbf{x}_i$. Eq. (20) is known as a *superposition of basis states*.

The *tensor product* of $|\mathbf{x}\rangle$ and $|\mathbf{y}\rangle$, represented by $|\mathbf{x}\rangle \otimes |\mathbf{y}\rangle$ and abbreviated as $|\mathbf{x}\rangle|\mathbf{y}\rangle$ or $|\mathbf{xy}\rangle$, is one form of putting together vector spaces to form larger vector spaces; it allows the combination of quantum states. Thus, the state space of a composite system is the tensor product of the state spaces of the component physical systems.

Two classical bits can be put together (combined) in a composite system as 00, 01, 10 and 11. The value of any combination can be written as the product of the individual bits. There are some quantum composite systems that cannot be written as a tensor product of states of its component systems, a property called *entanglement*. The Hilbert space of a system composed of two systems A and B is $H_A \otimes H_B$; that is, the tensor product of the respective spaces. Assuming that the first system is in an arbitrary state $|\mathbf{x}\rangle_A$ and the second system in another arbitrary state $|\mathbf{y}\rangle_B$, if the state of the composite system cannot be written as the tensor product $|\mathbf{x}\rangle_A \otimes |\mathbf{y}\rangle_B$, then the states are said to be entangled. The general state of a composite system $H_A \otimes H_B$ has the form

$$\Sigma_{\mathbf{x},\mathbf{y}} \mathbf{c}_{\mathbf{xy}} |\mathbf{x}\rangle_A |\mathbf{y}\rangle_B. \tag{21}$$

Examples of famous entangled two-qubit states are the Bell states:

$$\begin{aligned}
\beta_{00} &= |\mathbf{x}_0\rangle = \frac{1}{\sqrt{2}}\big(|00\rangle_{AB} + |11\rangle_{AB}\big),\\
\beta_{01} &= |\mathbf{x}_1\rangle = \frac{1}{\sqrt{2}}\big(|01\rangle_{AB} + |10\rangle_{AB}\big),\\
\beta_{10} &= |\mathbf{x}_2\rangle = \frac{1}{\sqrt{2}}\big(|00\rangle_{AB} - |11\rangle_{AB}\big),\\
\beta_{11} &= |\mathbf{x}_3\rangle = \frac{1}{\sqrt{2}}\big(|01\rangle_{AB} - |10\rangle_{AB}\big).
\end{aligned} \tag{22}$$

The *evolution* of a quantum system corresponds to how it changes over time. If the quantum system is *closed*, its evolution can be described by a *unitary transformation* or *operator* represented as a matrix. In other words, the state $|\mathbf{x}_2\rangle$ of the system at time t_2 is related to the state $|\mathbf{x}_1\rangle$ at time t_1 by a unitary transformation $\mathbf{A}$:

$$|\mathbf{x}_2\rangle = \mathbf{A}|\mathbf{x}_1\rangle. \tag{23}$$

The effect of a unitary transformation $\mathbf{A}$ on a state $\mathbf{x}$ is described by the corresponding rotation of the vector $|\mathbf{x}\rangle$ in the appropriate Hilbert space.

Examples of unitary transformations are the Pauli matrices:

$$\begin{aligned}
\sigma_0 &= \mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad \sigma_1 = \mathbf{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},\\
\sigma_2 &= \mathbf{Y} = \begin{bmatrix} 0 & i \\ -i & 0 \end{bmatrix}, \qquad \sigma_3 = \mathbf{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.
\end{aligned} \tag{24}$$

5.2.2. From classical to quantum computation

In computing and information theory, the most basic information unit is the *binary digit* or *bit*, which is an abstract entity that can assume either the value '0' or '1'. Information is physically stored as bits in standard computers, and the devices realizing a bit can be a combination of transistors and other integrated circuit elements with a distribution of charge determining the state of the bit. Bits can be combined into n-dimensional bitstrings to represent more information, and the bitstrings can be manipulated to perform computation; for instance, implement algorithms and perform mathematical operations. It is possible to access a certain memory address of a standard computer and observe (read) it without affecting its contents.

When information is stored at atomic scales, quantum effects take place and the result is a completely different scenario. In this case, a *quantum bit*, or *qubit* for short, can assume both values '0' and '1' simultaneously. Measurements and manipulations affect the contents of a qubit and can be modeled as matrix operations. Quantum information is described by a state in a two-level quantum mechanical system with two basic states conventionally labeled $|0\rangle$ and $|1\rangle$. These particular states form an orthonormal basis for a two-dimensional complex vector space (a Hilbert space H^2) and, thus, are known as *computational basis states*. As such, it is possible to form linear combinations of the

basis states, often called *superpositions*, which correspond to a *pure* qubit state:

$$|\mathbf{x}\rangle = \mathbf{c}_1|0\rangle + \mathbf{c}_2|1\rangle \tag{25}$$

where $\mathbf{c}_1$ and $\mathbf{c}_2$ are complex numbers.

In practice, the quantum superposition of the basis states means that an infinite amount of information can potentially be encoded in a single qubit by appropriately defining the amplitudes $\mathbf{c}_1$ and $\mathbf{c}_2$.

In standard computing, n bits can be used to represent 2^n different states. For instance, a string of 2 bits allows the states 00, 01, 10, and 11. Correspondingly, an n-qubit system has n computational basis states denoted as

$$|\mathbf{y}\rangle = \sum_{\mathbf{x}=00\ldots0}^{11\ldots1} \mathbf{c}_\mathbf{x}|\mathbf{x}\rangle \tag{26}$$

where $\mathbf{c}_\mathbf{x}$ are complex numbers such that $\sum_\mathbf{x} |\mathbf{c}_\mathbf{x}|^2 = 1$. Therefore, n qubits can represent any complex unit vector in a Hilbert space of dimension 2^n, having one dimension for each classical state. An ordered system of n qubits is also known as a *quantum register*.

The elementary operations that can be used to manipulate bits in standard computers are called *logic functions* (*operations*). A *logic gate* is an electronic device used to perform a simple logic function and to construct larger logic systems. A logic gate may have one or more inputs and a single bit output. Common gates are the NOT, which operates on a single bit, and the AND and OR gates that operate on two or more inputs.

There are small, well-defined sets of standard logic gates considered *universal*, meaning that any function can be performed by combining these universal gates. Examples are the NAND gate, the {AND, OR, NOT} set of gates, the {AND, NOT} gates, and others. Most classical gates, such as the OR and XOR gates, have the property that the operations they perform on the inputs are not invertible. This is easy to see by observing that one cannot say for sure what were the input values to these gates by having only their outputs and knowing the transformation they perform on the inputs. This is not the case for *quantum gates*.

Quantum gates, the analogues of logic gates in standard computers, are the basic units of quantum algorithms and, thus, quantum computers. The simplest example of a quantum gate acting on a single bit is the quantum NOT gate. While the classical NOT gate maps $1 \to 0$ and $0 \to 1$, the quantum NOT gate maps the state $|0\rangle \to |1\rangle$ and $|1\rangle \to |0\rangle$. This operation is suitably represented by matrix $\mathbf{X}$ defined as follows:

$$\begin{aligned}
\text{NOT}|0\rangle = \mathbf{X}|0\rangle &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle, \\
\text{NOT}|1\rangle = \mathbf{X}|1\rangle &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle.
\end{aligned} \tag{27}$$

Any unitary matrix transformation may be used as a quantum gate, what implies that, in contrast to standard gates, quantum gates are *reversible* (*invertible*). The Pauli matrices, presented in Eq. (24), correspond to well-known quantum gates. For instance, matrix $\mathbf{I}$ is the so-called *identity gate* and matrix $\mathbf{X}$ is the *NOT gate* presented above.

Another transformation that is commonly used in quantum computing is the *Hadamard gate*

$$\mathbf{H} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{28}$$

When applied to n bits individually, the Hadamard gate $\mathbf{H}$ generates a superposition of all 2^n possible states. For example, when applied to the basis states, the Hadamard gate creates a superposition of states.

$$\begin{aligned}
\mathbf{H}|0\rangle &= \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}\big(|0\rangle + |1\rangle\big), \\
\mathbf{H}|1\rangle &= \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}}\big(|0\rangle - |1\rangle\big).
\end{aligned} \tag{29}$$

Graphical descriptions of a quantum transformation, called *quantum circuits*, are a useful means of representing and combining quantum state transformations (quantum gates). The notation is similar to that of many layouts of classical circuits: information flow is from left to right; horizontal lines represent wires; vertical lines connecting two

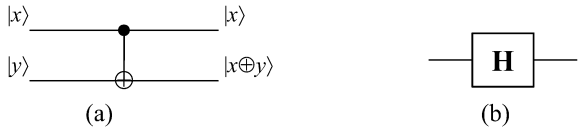

Fig. 9. Quantum circuits. (a) Controlled-NOT gate. (b) Hadamard gate.

or more qubits represent coupling terms; initial conditions are represented by quantum states written to the left of the circuit; and squared boxes together with special symbols placed on the horizontal lines correspond to gates. To illustrate, consider the quantum circuit associated with a gate called *controlled-NOT gate* (Fig. 9(a)). In the controlled-NOT gate circuit, the top wire is associated with the control qubit and the bottom wire is related to the target qubit. Some particular gates can be represented by a simple circuit composed of input/output wires and an appropriately labeled box, for instance, the Hadamard gate can be represented by a box with an **H** inside (Fig. 9(b)).

5.2.3. Scope of quantum computing

Similarly to DNA computing, quantum computing has been shown capable of speeding up several problem solving algorithms. For instance, Shor [255] proposed a quantum search algorithm for factoring that is exponentially faster than any classical algorithm known to date, and Grover [125] developed a quantum algorithm for search that is polinomially faster than any classical algorithm. But the importance and usefulness of quantum systems go beyond the speeding up of some information processing tasks. In quantum communication, for instance, qualitative and quantitative improvements can be gained, such as the reduction of the amount of the communicated data required. Furthermore, there are some tasks that can be performed in quantum systems, but that cannot be accomplished using classical physical systems. An example of such a task is quantum cryptography, which provides an absolute secrecy of communication.

5.2.4. Current trends and open problems

The main current trends and open problems in quantum computing are 'entangled'. These include quantum cryptography [117,261], the technology with which to build a quantum computer, i.e., the physical realization of quantum computers [80,153,162], investigations into quantum error correction and fault tolerance [47,111,258], the design of quantum programming languages [115,250], the development of novel quantum algorithms [51,259], and investigations into the power of quantum computing [139,168,204].

6. When natural computing should be used

There is a diverse range of problems to be solved, phenomena to be synthesized and questions to be answered. In all cases, one or more of the natural computing approaches, briefly overviewed here, will be used to solve the problem, synthesize the phenomenon or answer the question. However, natural computing approaches are not the only tools that provide solutions to these, nor are they always the most suitable and efficient approaches. In many situations there are alternative solution techniques for a given problem and these may even provide superior results. Thus, it is important to carefully investigate the problem to be solved before choosing a specific solution method, be it a natural computing tool or a different one. To give some hints, natural computing should be used when:

- The problem to be solved is complex, i.e., involves a large number of variables or potential solutions, is highly dynamic, nonlinear, has multiple objectives, etc.
- It is not possible to guarantee that a potential solution found is optimal, but it is possible to find a quality measure that allows the comparison of solutions among themselves.
- The problem to be solved cannot be (suitably) modeled, such as pattern recognition and classification (e.g., vision) tasks. In some cases, although it is not possible to model the problem, there are examples (samples) available that can be used to 'teach' the system how to solve the problem, and the system is somehow capable of 'learning from examples'.
- A single solution is not good enough or when diversity is important. Most standard problem-solving techniques are able to provide a single solution to a given problem, but are not capable of providing more than one solution.

One reason for that is because most standard techniques are deterministic and natural computing is, in its majority, composed of stochastic methods.

- Biological, physical, and chemical systems and processes have to be synthesized with realism. Euclidean geometry is very good and efficient to create man-made forms, but has difficulty in reproducing natural patterns. This is because nature is fractal, and fractal geometry provides the most suitable tools with which to synthesize natural patterns.
- Life behaviors and phenomena have to be synthesized in artificial media. No matter the artificial media (e.g., a computer or a robot), the essence of a given natural behavior or pattern is extracted and synthesized in a, usually, much simpler form in artificial life systems.
- The limits of current technology are reached or new computing materials have to be sought. Nature abounds with information storage and processing systems, and the scientific and engineering aspects of how to use these natural materials to compute are the main challenges of the third branch of natural computing: computing with natural materials.

7. Conclusions

Natural computing has taught us to *think 'naturally' about computation* and also to think *computationally about nature*. All natural computing approaches yield novel and exciting capabilities for science as a whole, mainly computer science, engineering, philosophy, and the biosciences. Scientists and engineers have been gifted with a very rich paradigm for exploration; the combination of nature and computing. In a broad sense, natural computing has taught us that any (model of a) natural phenomenon may be used as a basis for the development of novel algorithmic tools for problem solving; a vast array of natural phenomena can be simulated/emulated in computers; and natural materials can be used for computation. The fruits of these explorations are continuously becoming new technological solutions and explanations to old and recent problems, and the full potential is far from being reached.

Over the last two decades, natural computing has experienced an astonishing growth. Its frontiers have been expanding in many directions simultaneously, from nature-inspired computing to the emerging computing with new natural materials. This rapid growth is due to a number of factors: (1) computing power and memory storage capacity have increased dramatically in these last decades; (2) the vast number of problems with features such as nonlinearity, high-dimensionality, difficulty in modeling and finding derivatives, etc., make unfeasible the use of traditional techniques, such as linear, nonlinear, dynamic, integer programming, and others; and (3) increasing interaction and collaboration among different disciplines and fields of investigation, like computer science, engineering, biology, medicine, physics, ecology, neuroscience, mathematics, and statistics. Despite the rapid growth of the field, it is important to be aware of when to use natural computing approaches, as discussed above.

Research in the area has been performed in many frontlines, from applications, to theoretical investigations, to physical realization. The applications of natural computing are manifold, and in the last years we have witnessed a growth in importance of the applications to optimization problems in general; biological applications and bioinformatics; applications in data mining (including web mining), games, arts, music, biometrics, robotics, economics and finance, forecasting problems, and applications involving the security of information systems. In theoretical terms, the investigation into formal aspects, such as convergence and scalability analysis, universal approximation/computation capabilities, the comparison with other methods (including classical ones), the design of novel operators, the automatic tuning (or guidelines for a better choice) of user-defined parameters, the proposal of theoretical frameworks with which to investigate and/or apply natural computing have not only gained special attention but also remain some of the main challenges of the field. The search for novel nature-inspired algorithms and the identification of problems to which each of the methods is most suitable for has been pursued, together with an understanding of why and how they work or not.

The interdisciplinary and multidisciplinary spirit fostered as a result of doing natural computing research is one of the most pleasant and remarkable features of this field. The excitement and freshness of the field build up the prospects for creativity, invention, innovation, and discovery in this endeavor. Most researchers of natural computing agree that the approaches described in this paper represent not only the most accessible and exploitable openings to a large set of computational systems—the opening wedge into a new field that is likely to provide a satisfying synergy between the worlds of Nature and Computing.

Acknowledgements

The author would like to thank CNPq and FAPESP for the financial support, and the reviewers for their comments and suggestions on previous versions of this manuscript.

References

[1] Adleman LM. Molecular computation of solutions to combinatorial problems. Science 1994;226:1021–4.
[2] Adleman LM, Rothemund PWK, Roweis S, Winfree E. On applying molecular computations to the data encryption standard. In: Baum E, Boneh D, Kaplan P, Lipton R, Reif J, Seeman N, editors. Proc 2nd annual meeting on DNA based computers. 1996. p. 28–48.
[3] Aickelin U, Cayzer S. The danger theory and its application to artificial immune systems. In: Timmis J, Bentley PJ, editors. Proceedings of the 1st international conference on artificial immune systems (ICARIS). 2002. p. 141-48.
[4] Aickelin U, Dasgupta D. Artificial immune systems tutorial. In: Burke E, Kendall G, editors. Search methodologies—introductory tutorials in optimization and decision support techniques. Kluwer; 2005. p. 375–99.
[5] Aickelin U, Bentley P, Cayzer S, Kim J, McLeod J. Danger theory: The link between AIS and IDS? Lecture notes in computer science, vol. 2787. 2003. p. 147–55.
[6] Aickelin U, Greensmith J, Twycross J. Immune system approaches to intrusion detection—A review. In: Nicosia G, Cutello V, Bentley PJ, Timmis J, editors. Proceedings of the 3rd international conference on artificial immune systems. Lecture notes in computer science, vol. 3239. Springer; 2004. p. 316–29.
[7] Allen D, et al. Timing, genetic requirements and functional consequences of somatic hypermutation during B-cell development. Imm Rev 1987;96:5–22.
[8] Amos M. DNA computation. PhD thesis. Department of Computer Science, The University of Warwick; 1997.
[9] Amos M. Theoretical and experimental DNA computation. Springer; 2005.
[10] Amos M, Paun G, Rozenberg G, Salomaa A. Topics in the theory of DNA computing. Theoretical Computer Science 2002;287(1):3–38.
[11] Amos M, Wilson S, Hodgson DA, Owenson G, Gibbons A. Practical implementation of DNA computations. In: Calude CS, Casti J, Dinneen MJ, editors. Unconventional models of computation. 1998. p. 1–18.
[12] Anderson RL. Recent advances in finding best operating conditions. J Amer Statist Assoc 1953;48:789–98.
[13] Arnold DV, Beyer H-G. Optimum tracking with evolution strategies. Evolutionary Computation 2006;14(3):291–308.
[14] Bäck T, Fogel DB, Michalewicz Z. Evolutionary computation 1 basic algorithms and operators. Bristol and Philadelphia: Institute of Physics Publishing (IOP); 2000.
[15] Bäck T, Fogel DB, Michalewicz Z. Evolutionary computation 2 advanced algorithms and operators. Bristol and Philadelphia: Institute of Physics Publishing (IOP); 2000.
[16] Baish JW, Jain RK. Fractals and cancer. Cancer Research 2000;60:3683–8.
[17] Banzhaf W, Foster J. Biological applications of genetic and evolutionary computation. Genetic Programming and Evolvable Machines 2004;5(2):119–241.
[18] Barnsley MF, Hurd LP. Fractal image compression. Boston: AK Peters; 1992.
[19] Barnsley MF. Fractal functions and interpolation. Journal Constructive Approximation 1986;2(1):303–29.
[20] Baum EB. Building an associative memory vastly larger than the brain. Science 1995;268:583–5.
[21] Beasley D. Possible applications of evolutionary computation. In: Bäck T, Fogel DB, Michalewicz Z, editors. Evolutionary computation 1 basic algorithms and operators. Bristol and Philadelphia: Institute of Physics Publishing (IOP); 2000. p. 4–19 [chapter 2].
[22] Bedau MA, McCaskill JS, Packard NH, Rasmussen S, Adami C, Green DG, Ikegami T, Kaneko K, Ray TS. Open problems in artificial life. Artificial Life 2000;VI:363–76.
[23] Beigel R, Fu B. Solving intractable problems with DNA computing. In: Proc of the 13th annual IEEE conference on computational complexity. 1998. p. 154–68.
[24] Beni G. The concept of cellular robotic systems. In: Proc of the IEEE Int. Symp. on Intelligent Control. 1988. p. 57–62.
[25] Beni G, Wang J. Swarm intelligence. In: Proc of the 7th annual meeting of the robotics society of Japan. 1989. p. 425–8.
[26] Bennett CH. Logical reversibility of computation. IBM Journal of Research and Development 1973;17:525–32.
[27] Bentley PJ. Evolutionary design by computers. Morgan Kaufmann; 1999.
[28] Bentley PJ, Corne DW. Creative evolutionary systems. Morgan Kaufmann; 2001.
[29] Beyer H-G. Theory of evolution strategies. Springer; 2001.
[30] Bilchev G, Parmee IC. The ant colony metaphor for searching continuous design spaces. Evolutionary Computing, AISB workshop 1995:25–39.
[31] Birattari M, Di Caro G, Dorigo M. Toward the formal foundation of ant programming. In: Dorigo M, Di Caro G, Sampels M, editors. Ant algorithms. Proc ANTS 2002. Third internat workshop. Lecture notes in computer science, vol. 2463. 2002. p. 188–201.
[32] Bishop CM. Neural networks for pattern recognition. Oxford University Press; 1996.
[33] Blackwell TM. Swarms in dynamic environments. In: Proceedings of the genetic and evolutionary computation conference 2003 (GECCO 2003), Chicago, IL, USA. Lecture notes in computer science, vol. 2723. 2003. p. 1–12.
[34] Blum C. Ant colony optimization: introduction and recent trends. Physics of Life Reviews 2005;2:353–73.
[35] Blum C, Dorigo M. The hyper-cube framework for ant colony optimization. IEEE Trans Syst Man Cybernet Part B 2004;34(2):1161–72.
[36] Bonabeau E, Théraulaz G. Swarm smarts. Scientific American 2000;March:72–9.
[37] Bonabeau E, Dorigo M, Théraulaz G. Inspiration for optimization from social insect behavior. Nature 2000;406:39–42.

[38] Bonabeau E, Dorigo M, Théraulaz G. Swarm intelligence: from natural to artificial systems. Oxford University Press; 1999.
[39] Boneh D, Dunworth C, Lipton RJ. Breaking the DES using a molecular computer. In: Lipton RJ, Baum EB, editors. DNA based computers—Proc of the DIMACS workshop 1995. 1996. p. 37–66.
[40] Bremermann HJ. Optimization through evolution and recombination. In: Yovits MC, Jacobi GT, Goldstein DG, editors. Self-organizing systems. Washington, DC: Spartan; 1962. p. 93–106.
[41] Brits R, Engelbrecht AP, van den Bergh F. Scalability of niche PSO. In: Proceedings of the IEEE swarm intelligence symposium 2003 (SIS 2003), Indianapolis, Indiana, USA. 2003. p. 228–34.
[42] Burges CJC. A tutorial on support vector machines for pattern recognition. Data Mining Knowledge Discovery 1998;2(2):1–47.
[43] Burnet FM. Clonal selection and after. In: Bell GI, Perelson AS, Pimbley Jr GH, editors. Theoretical immunology. Marcel Dekker; 1978. p. 63–85.
[44] Calude CS, Păun G. Computing with cells and atoms. Taylor & Francis; 2001.
[45] Cazangi RR, Von Zuben FJ, Figueiredo MF. Stigmergic autonomous navigation in collective robotics. In: Abraham A, Grosan C, Ramos V, editors. Stigmergic optimization. Berlin: Springer; 2006. p. 25–64.
[46] Chand AKB, Kapoor GP. Generalized cubic spline fractal interpolation functions. SIAM Journal on Numerical Analysis 2006;44(2):655–76.
[47] Chiaverini J, Leibfried D, Schaetz T, Barrett MD, Blakestad RB, Britton J, Itano WM, Jost JD, Knill E, Langer C, Ozeri R, Wineland DJ. Realization of quantum error correction. Nature 2004;432:602–5.
[48] Christiansen M, Kirby S. Language and evolution: consensus and controversies. Trends in Cognitive Science 2003;7(7):300–7.
[49] Churchland P, Sejnowski TJ. The computational brain. MIT Press; 1992.
[50] Clerc M, Kennedy J. The particle swarm explosion, stability, and convergence in a multidimensional complex space. IEEE Trans on Evolutionary Computation 2002;6(1):58–73.
[51] Cleve R, Ekert A, Macchiavello C, Mosca M. Quantum algorithms revisited. Proceedings of the Royal Society of London A 1998;454:339–54.
[52] Coath G, Halgamuge SK. A comparison of constraint-handling methods for the application of particle swarm optimization to constrained nonlinear optimization problems. In: Proceedings of IEEE congress on evolutionary computation 2003 (CEC 2003), Canbella, Australia. 2003. p. 2419–25.
[53] Cohen N. Fractal antenna applications in wireless telecommunications. IEEE Proc Professional Program Electronics Industry Forum 1997:43–9.
[54] Conrad M. Information processing in molecular systems. Currents in Modern Biology 1972;5:1–14.
[55] Corne DW, Bentley PJ. Creative evolutionary systems. Morgan Kaufmann; 2001.
[56] Cortes C, Vapnik V. Support vector networks. Machine Learning 1995;20:273–97.
[57] Cristianini N, Shawe-Taylor J. An introduction to support vector machines and other kernel-based learning methods. Cambridge University Press; 2000.
[58] Cross SS. Fractals in pathology. Journal of Pathology 1997;182:1–8.
[59] Cutello V, Nicosia G. The clonal selection principle for in silico and in vitro computing. In: de Castro LN, Von Zuben FJ, editors. Recent developments in biologically inspired computing. Idea Group Publishing; 2004. p. 104–46 [chapter VI].
[60] Czarn A, MacNish C, Vijayan K, Turlach B, Gupta R. Statistical exploratory analysis of genetic algorithms. IEEE Trans on Evolutionary Computation 2004;8(4):405–21.
[61] Dasgupta D, editor. Artificial immune systems and their applications. Springer; 1999.
[62] Dasgupta D, Attoh-Okine N. Immunity-based systems: a survey. Proc of the IEEE SMC 1997;1:369–74.
[63] Dasgupta D, Michalewicz Z. Evolutionary algorithms in engineering applications. Springer; 1997.
[64] Dasgupta D, Majumdar N, Niño F. Artificial immune systems: A bibliography, CS Technical report, CS-01-002. 2001 [on-line] http://www.cs.memphis.edu/~dasgupta/AIS/AIS-bib.pdf.
[65] d'Avila Garcez A, Lamb LC. A connectionist computational model for epistemic and temporal reasoning. Neural Computation 2006;18(7):1711–38.
[66] Dayan P, Abbot LF. Theoretical neuroscience: computational and mathematical modeling of neural systems. MIT Press; 2001.
[67] de Castro LN. Immune cognition, micro-evolution, and a personal account on immune engineering. SEED Journal (Semiotics, Evolution, Energy, and Development), University of Toronto 2003;3(3):134–55.
[68] de Castro LN. Engineering applications of artificial immune systems. Talk presented at ICARIS 2004. 2004.
[69] de Castro LN. Natural computing. In: Khosrow-Pour M, editor. Encyclopedia of information science and technology, vol. IV. Idea Group Inc; 2005. p. 2080–4.
[70] de Castro LN. Fundamentals of natural computing: basic concepts, algorithms, and applications. CRC Press LLC; 2006.
[71] de Castro LN. Artificial immune systems: the past, the present. and the future?" Tutorial presented at the 5th international conference on artificial immune systems, Oeiras, Portugal, September 2006.
[72] de Castro LN, Timmis JI. Artificial immune systems: a new computational intelligence approach. Springer; 2002.
[73] de Castro LN, Von Zuben FJ. Artificial immune systems: Part II—A survey of applications. Technical Report—RT DCA 02/00. 2000. p. 65. [on-line] http://www.dca.fee.unicamp.br/~lnunes.
[74] de Castro LN, Von Zuben FJ. aiNet: an artificial immune network for data analysis. In: Abbas HA, Sarker RA, Newton CS, editors. Data mining: a heuristic approach. USA: Idea Group Publishing; 2001. p. 231–59 [chapter XII].
[75] de Castro LN, Von Zuben FJ. Learning and optimization using the clonal selection principle. IEEE Transactions on Evolutionary Computation 2002;6(3):239–51.
[76] de Castro LN, Von Zuben FJ. Recent developments in biologically inspired computing. Idea Group Publishing; 2004.
[77] Deaton R, Garzon M, Rose JA, Murphy RC, Stevens Jr, SE, Franceschetti DR. A DNA based artificial immune system for self-nonself discrimination. In: Proc of the IEEE SMC'97. 1997. p. 862–6.

[78] Deneubourg J-L, Aron S, Goss S, Pasteels JM. The self-organizing exploratory pattern of the Argentine ants. J Insect Behavior 1990;3:159–68.
[79] Dirac P. The principles of quantum mechanics. 4th ed. Oxford University Press; 1982.
[80] DiVincenzo DP. The physical implementation of quantum computation. Fortschr Phys 2000;48:771–83.
[81] Dorigo M. Optimization, learning and natural algorithms. PhD thesis. Dipartimento di Elettronica, Politecnico di Milano, Italy; 1992 [in Italian].
[82] Dorigo M, Blum C. Ant colony optimization theory: A survey. Theoretical Computer Science 2005;344(2–3):243–78.
[83] Dorigo M, Di Caro G. The ant colony optimization meta-heuristic. In: Corne D, Dorigo M, Glover F, editors. New ideas in optimization. McGraw-Hill; 1999. p. 13–49.
[84] Dorigo M, Stützle T. Ant colony optimization. MIT Press; 2004.
[85] Dorigo M, Di Caro G, Gambardella LM. Ant algorithms for discrete optimization. Artificial Life 1999;5(3):137–72.
[86] Dorigo M, Maniezzo V, Colorni A. Ant system: Optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybernet Part B 1996;26(1):29–41.
[87] Dorigo M, Maniezzo V, Colorni A. The ant system: Optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybernet Part B 1996;26(1):29–41.
[88] Emmeche C. Closure, function, emergence, semiosis and life: the same idea? Reflections on the concrete and the abstract in theoretical biology". In: Chandler JLR, Van de Vijver G, editors. Closure: emergent organizations and their dynamics. Annals of the New York Academy of Sciences, vol. 901. New York: The New York Academy of Sciences; 2000. p. 187–97.
[89] Emmeche C, Koppe S, Stjernfelt F. Explaining emergence: towards an ontology of levels. Journal for General Philosophy of Science 1997;28:83–119.
[90] Engelbrecht. Fundamentals of computational swarm intelligence. John Wiley & Sons; 2006.
[91] Esponda F, Forrest S, Helman P. A formal framework for positive and negative detection schemes. IEEE Trans Syst Man Cybernet Part B 2004;34(1):357–73.
[92] Ezziane Z. DNA computing: applications and challenges. Nanotechnology 2006;17(2):R27–39.
[93] Falconer K. Fractal geometry: mathematical foundations and applications. John Wiley & Sons; 2003.
[94] Farina M, Deb K, Amato P. Dynamic multiobjective optimization problems: test cases, approximations, and applications. IEEE Trans Evolutionary Computation 2004;8(5):425–42.
[95] Farmer JD, Belin Ad'A. Artificial life: the coming evolution. In: Langton C, Taylor C, Farmer JD, Rasmussen S, editors. Artificial life II. 1991. p. 815–40.
[96] Farmer JD, Packard NH, Perelson AS. The immune system, adaptation, and machine learning. Physica D 1986;22:187–204.
[97] Fausett L. Fundamentals of neural networks: architectures, algorithms, and applications. Prentice-Hall; 1994.
[98] Firaxis Games. Civilization IV. 2005; http://www.2kgames.com/civ4/home.htm.
[99] Flake GW. The Computational beauty of nature. MIT Press; 2000.
[100] Fogel DB. Evolutionary computation: toward a new philosophy of machine intelligence. 2nd ed. The IEEE Press; 2000.
[101] Fogel DB. Evolutionary computation: principles and practice for signal processing. SPIE—The International Society for Optical Engineering; 2000.
[102] Fogel DB, editor. Evolutionary computation: the fossil record. The IEEE Press; 1998.
[103] Fogel GB, Corne DW. Evolutionary computation in bioinformatics. Morgan Kaufmann; 2002.
[104] Forrest S, Perelson A, Allen L, Cherukuri R. Self-nonself discrimination in a computer. In: Proc of the IEEE symposium on research in security and privacy. 1994. p. 202–12.
[105] Forrest S, Javornik B, Smith RE, Perelson AS. Using genetic algorithms to explore pattern recognition in the immune system. Evolutionary Computation 1993;1(3):191–211.
[106] Fraser AS. Simulation of genetic systems by automatic digital computers. Aust J Biol Sci 1959;10:489–99.
[107] Freitas AA, Rozenberg G. Data mining and knowledge discovery with evolutionary algorithms. Springer; 2002.
[108] Freitas AA, Timmis J. Revisiting the foundations of artificial immune systems: a problem-oriented perspective. Lecture notes in computer science, vol. 2787. 2003. p. 229–41.
[109] Friedberg RM. A learning machine: part I. IBM J Research and Development 1958;2:2–13.
[110] Futuyma DJ. Evolutionary biology. 3rd ed. Sinauer Associates; 1998.
[111] Gaitan F. Quantum error correction and fault tolerant computing. Wiley; 2007.
[112] Galeano JC, Veloza-Suan A, González FA. A comparative analysis of artificial immune network models. In: Proc of the genetic and evolutionary computation conference. 2005. p. 361–8.
[113] Garey MR, Johnson D. Computers and intractability: a guide to the theory of NP-completeness. Freeman; 1979.
[114] Garrett S. How do we evaluate artificial immune systems. Evolutionary Computation 2005;13(2):145–78.
[115] Gay SJ. Quantum programming languages: survey and bibliography. Mathematical Structures in Computer Science 2006;16(4):581–600.
[116] Gehani A, LaBean TH, Reif JH. DNA-based cryptography. In: Jonoska N, Paun G, Rozenberg G, editors. Aspects of molecular computing, essays dedicated to tom head on the occasion of his 70th birthday. Lecture notes in computer science, vol. 2950. 2004. p. 167–88.
[117] Gisin N, Ribordy G, Tittel W, Zbinden H. Quantum cryptography. Review of Modern Physics 2002;74(1):145–95.
[118] Goldberg DE. Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley; 1989.
[119] Golden RM. Mathematical methods for neural network analysis and design. Cambridge, MA: MIT Press; 1996.
[120] González F, Dasgupta D. Anomaly detection using real-valued negative selection. Genetic Programming and Evolvable Machines 2003;4(4):383–403.
[121] Goss S, Aron S, Deneubourg JL, Pasteels JM. Self organized shortcuts in the Argentine ants. Naturwissenchaften 1989;76:579–81.

[122] Goss S, Beckers R, Deneubourg JL, Aron S, Pasteels JM. How trail laying and trail following can solve foraging problems for ant colonies. In: Hughes RN, editor. Behavioural mechanisms of food selection. NATO–ASI series, vol. G20. Springer; 1990. p. 661–78.
[123] Gramß T, Bornholdt S, Groß M, Mitchell M, Pellizzari T. Non-standard computation: molecular computation—cellular automata—evolutionary algorithms—quantum computers. Wiley-VCH; 2001.
[124] Greensmith J, Aickelin U, Cayzer S. Introducing dendritic cells as a novel immune-inspired algorithm for anomaly detection. Lecture notes in computer science, vol. 3627. 2005. p. 153–67.
[125] Grover LK. Quantum mechanics helps in searching for a needle in a haystack. Physical Review Letters 1997;79(2):325–8.
[126] Guarnieri F, Fliss M, Bancroft C. Making DNA add. Science 1996;273(5272):220–3.
[127] Guozhen X, Mingxin L, Lei Q, Xuejia L. New field of cryptography: DNA cryptography. Chinese Science Bulletin 2006;51(12):1413–20.
[128] Gutjahr WJ. ACO algorithms with guaranteed convergence to the optimal solution. Information Processing Letters 2002;82(3):145–53.
[129] Hagiya M, Rose JA, Komiya K, Sakamoto K. Complexity analysis of the SAT engine: DNA algorithms as probabilistic algorithms. Theoretical Computer Science 2002;287(1):59–71.
[130] Hahn HK, Evertsz CJG, Fasel JHD, Peitgen HO. Fractal properties segment anatomy and interdependence of the human portal vein and the hepatic vein in 3D. Fractals 2003;11:53–62.
[131] Hammer B, Micheli A, Sperduti A. Universal approximation capability of cascade correlation for structures. Neural Computation 2005;17(5):1109–59.
[132] Hart E, Timmis J. Application areas of AIS: the past, the present and the future. Lecture notes in computer science, vol. 3627. 2005. p. 483–98.
[133] Havlin S, Buldyrev SV, Goldberger AL, Mantegna RN, Ossadnik SM, Peng CK, Simon M, Stanley HE. Fractals in biology and medicine. Chaos Solitons & Fractals 1995;6:171–201.
[134] Haykin S. Neural networks: a comprehensive foundation. 2nd ed. Prentice-Hall; 1999.
[135] Head T. Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. Bulletin of Mathematical Biology 1987;49:737–59.
[136] Higashi N, Iba H. Particle swarm optimization with Gaussian mutation. In: Proceedings of the IEEE swarm intelligence symposium 2003 (SIS 2003), Indianapolis, Indiana, USA. 2003. p. 72–9.
[137] Hightower RR, Forrest SA, Perelson AS. The evolution of emergent organization in immune system gene libraries. In: Eshelman LJ, editor. Proc of the 6th Int conference on genetic algorithms. Morgan Kaufmann; 1995. p. 344–50.
[138] Hirvensalo M. Quantum computing. Springer; 2000.
[139] Hirvensalo M. Computing with quanta—impacts of quantum theory on computation. Theoretical Computer Science 2002;287(2002):267–98.
[140] Hoai NX, McKay RI, Essam D. Representation and structural difficulty in genetic programming. IEEE Trans on Evolutionary Computation 2006;10(2):157–66.
[141] Hofmeyr SA, Forrest S. Architecture for an artificial immune system. Evolutionary Computation 2000;7(1):45–68.
[142] Holland JJ. Adaptation in natural and artificial systems. MIT Press; 1975.
[143] Hruschka ER, Ebecken N. Extracting rules from multilayer perceptrons in classification problems: a clustering-based approach. Neurocomputing, 2006. In press.
[144] Huang G-B, Saratchandran P, Sundararajan N. A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation. IEEE Transactions on Neural Networks 2005;16(1):57–67.
[145] Iannaccone PM, Khokha M, editors. Fractal geometry in biological systems. CRC Press; 1996.
[146] Ilachinski A. Cellular automata: a discrete universe. World Scientific; 2001.
[147] Ishida Y. The immune system as a self-identification process: a survey and a proposal. In: Proc of the ICMAS Int workshop on immunity-based systems. 1996. p. 2–12.
[148] Jacob C, Pilat ML, Timmis J, Bentley PJ. In: Proceedings of the 4th international conference on artificial immune systems. Lecture notes in computer science, vol. 3627. Springer; 2005.
[149] Jacobsson H. Rule extraction from recurrent neural networks: a taxonomy and review. Neural Computation 2005;17(6):1223–63.
[150] Janeway CA, Travers P, Walport M, Capra JD. Immunobiology: the immune system in health and disease. 4th ed. Garland Publishing; 1999.
[151] Jerne NK. Towards a network theory of the immune system. Ann Immunol (Inst Pasteur) C 1974;125:373–89.
[152] Johnson S. Emergence: the connected lives of ants, brains, cities and software. Penguin Books; 2002.
[153] Kanamori Y, Yoon S-M, Pan WD, Sheldon FT. A short survey on quantum computers. International Journal of Computers and Applications 2006;28(3).
[154] Karpenko O, Shi J, Dai Y. Prediction of MHC class II binders using the ant colony search strategy. Artificial Intelligence in Medicine 2005;35(1–2):147–56.
[155] Karr CL, Freeman LM. Industrial applications of genetic algorithms. CRC Press; 1998.
[156] Keeley BL. Evaluating artificial life and artificial organisms. In: Langton C, Shimohara K, editors. Artificial life V. Addison-Wesley; 1997. p. 264–71.
[157] Kelsey J, Timmis J. Immune inspired somatic contiguous hypermutation for function optimisation. In: Cantu-Paz, et al., editors. Proc of the genetic and evolutionary computation conference. Lecture notes in computer science, vol. 2723. Springer; 2003. p. 207–18.
[158] Kennedy J. The particle swarm: social adaptation of knowledge. In: Proc of the IEEE Int Conf on evolutionary computation. 1997. p. 303–8.
[159] Kennedy J. Particle swarms: optimization based on sociocognition. In: de Castro LN, Von Zuben FJ, editors. Recent developments in biologically inspired computing. Idea Group Publishing; 2004. p. 235–69 [chapter X].
[160] Kennedy J, Eberhart R. Particle swarm optimization. In: Proc of the IEEE Int Conf on neural networks, Perth, Australia, 4. 1995. p. 1942–8.
[161] Kennedy J, Eberhart R, Shi Y. Swarm intelligence. Morgan Kaufmann; 2001.
[162] Keyes RW. Challenges for quantum computing with solid-state devices. IEEE Computer 2005;38(1):65–9.
[163] Kim K, Yoon S-M. Dynamical behavior of continuous tick data in futures exchange market. Fractals 2003;11:131–6.

[164] Kim S. Player's positional dependence of fractal behaviors in a soccer game. Fractals 2006;14(1):71–6.
[165] Kirby S. Natural language from artificial life. Artif Life 2002;VIII:185–215.
[166] Knidel H, de Castro LN, Von Zuben FJ. Data clustering with a neuro-immune network. Lecture notes in computer science, vol. 3627. 2005. p. 1279–88.
[167] Kohonen T. Self-organizing maps. Springer; 2000.
[168] Konig R, Maurer U, Renner R. On the power of quantum memory. IEEE Transactions on Information Theory 2005;51(7):2391–401.
[169] Koza JR. Genetic programming: on the programming of computers by means of natural selection. MIT Press; 1992.
[170] Koza JR. Genetic programming II: automatic discovery of reusable programs. MIT Press; 1994.
[171] Kube CR, Parker CAC, Wang T, Zhang H. Biologically inspired collective robotics. In: de Castro LN, Von Zuben FJ, editors. Recent developments in biologically inspired computing. Idea Group Publishing; 2004. p. 367–97 [chapter XV].
[172] Kurnaz ML. Application of detrended fluctuation analysis to monthly average of the maximum daily temperatures to resolve different climates. Fractals 2004;12(4):365–73.
[173] Kushchu I. Web-based evolutionary and adaptive information retrieval. IEEE Trans Evolutionary Computation 2005;9(2):117–25.
[174] Kwok TY, Yeung DY. Constructive algorithms for structure learning in feedforward neural networks for regression problems. IEEE Trans on Neural Networks 1997;8(3):630–45.
[175] Langdon WB, Gustafson S. Genetic programming and evolvable machines: five years of reviews. Genetic Programming and Evolvable Machines 2005;6(2):221–8.
[176] Langton CG. Artificial life. In: Langton C, editor. Artificial life. Addison-Wesley; 1988. p. 1–47.
[177] Lesmoir-Gordon N, Rood W, Edney R. Introducing fractal geometry. ICON Books UK; 2000.
[178] Lindenmayer A. Mathematical models for cellular interaction in development, parts I and II. Journal of Theoretical Biology 1968;18:280–315.
[179] Loocke PV. Visualization of data on basis of fractal growth. Fractals 2004;12(1):123–36.
[180] Ma L, Khorasani K. Constructive feedforward neural networks using Hermite polynomial activation functions. IEEE Trans Neural Networks 2005;16(4):821–33.
[181] Maley CC. DNA computation: theory, practice, and prospects. Evolutionary Computation 1998;6(3):201–29.
[182] Maley CC. Four steps toward open-ended evolution. In: Banzhaf W, Daida J, Eiben AE, Garzon MH, Honavar V, Jakiela M, Smith RE, editors. Proceedings of the genetic and evolutionary computation conference. Morgan Kaufmann; 1999. p. 1336–43.
[183] Mandelbrot B. The fractal geometry of nature. WH Freeman and Company; 1983.
[184] Mandelbrot BB. Fractals and scaling in finance. Springer; 2005.
[185] Mandelbrot BB, Hudson RL. The (Mis)behavior of markets—a fractal view of risk, ruin, and reward. Basic Books; 2004.
[186] Mange D, Tomassini M. Bio-inspired computing machines: towards novel computational architecture. Presses Polytechniques et Universitaires Romandes; 1998.
[187] Mannie MD. Immunological self/nonself discrimination. Immunological Research 1999;19(1):65–87.
[188] Margulis L, Sagan D, Eldredge N. What is life? University of California Press; 2000.
[189] Matia K, Ashkenazy Y, Stanley HE. Multifractal properties of price fluctuations of stock and commodities. Europhysics Letters 2003;61:422–8.
[190] Maxis. SimCity 4. http://simcity.ea.com/, 2003.
[191] McClelland JL, Rumelhart DE, The PDP Research Group. Parallel distributed processing: explorations in the microstructure of cognition, vol. 2: psychological and biological models. MIT Press; 1986.
[192] McCulloch W, Pitts WH. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics 1943;5:115–33.
[193] Meuleau N, Dorigo M. Ant colony optimization and stochastic gradient descent. Artificial Life 2002;8(2):103–21.
[194] Michalewicz Z. Genetic algorithms + data structures = evolution programs. Springer; 1996.
[195] Mills Jr. AP, Turberfield M, Turberfield AJ, Yurke B, Platzman PM. Experimental aspects of DNA neural network computation. Soft Computing 2001;5(1):10–8.
[196] Mitchell M. An introduction to genetic algorithms. MIT Press; 1996.
[197] Mulawka JJ, Borsuk P, Weglenski P. Implementation of the inference engine based on molecular computing technique. In: Proc of the IEEE Int Conf on evolutionary computation (ICEC'98). 1998. p. 493–8.
[198] Na D, Lee D. Mathematical modeling of immune suppression. Lecture notes in computer science, vol. 3627. 2005. p. 182–92.
[199] Na D, Park I, Lee KH, Lee D. Integration of immune models using petri nets. Lecture notes in computer science, vol. 3239. 2004. p. 205–16.
[200] Navascués MA, Sebastián MV. Generalization of Hermite functions by fractal interpolation. Journal of Approximation Theory 2004;131(1):19–29.
[201] Neal M. Meta-stable memory in an artificial immune network. In: Timmis J, Bentley P, Hart E, editors. Proc of the 2nd international conference on artificial immune systems. Lecture notes in computer science, vol. 2787. Springer; 2003. p. 168–80.
[202] Nicollet M, Lemarchand A, Cavaciuti N. Detection of atmospheric turbulence by multifractal analysis using wavelets. Fractals 2004;12(2):211–21.
[203] Nicosia G, Cutello V, Bentley PJ, Timmis J. In: Proceedings of the 3rd international conference on artificial immune systems. Lecture notes in computer science, vol. 3239. Springer; 2004.
[204] Nielsen MA, Chuang IL. Quantum computation and quantum information. Cambridge University Press; 2000.
[205] Nolfi S, Floreano D. Evolutionary robotics: the biology, intelligence, and technology of self-organizing machines. MIT Press/Bradford Books; 2000.
[206] Nolfi S, Floreano D. Synthesis of autonomous robots through evolution. Trends in Cognitive Science 2002;6:31–7.
[207] O'Neill B. Digital evolution. PLoS Biol 2003;1(1):e8.

[208] O'Neill M, Ryan C. Grammatical evolution: evolutionary automatic programming in an arbitrary language. Kluwer Academic; 2003.
[209] O'Reilly RC, Munakata Y. Computational explorations in cognitive neuroscience: understanding the mind by simulating the brain. MIT Press; 2000.
[210] Ofria C, Wilke CO. Avida: a software platform for research in computational evolutionary biology. Artificial Life 2004;X:191–229.
[211] Oliver JS. Computation with DNA: matrix multiplication. In: Landweber L, Baum E, editors. DNA Based Computers II. Proc of DIMACS, vol. 44. 1998. p. 113–22.
[212] Oprea M, Forrest S. Simulated evolution of antibody gene libraries under pathogen selection. In: Proc of the IEEE System, Man, and Cybernetics. 1998.
[213] Ouyang Q, Kaplan PD, Liu S, Libchaber A. DNA solution of the maximal clique problem. Science 1997;278:446–9.
[214] Pang W, Wang K-P, Zhou C-G, Dong L-J. Fuzzy discrete particle swarm optimization for solving traveling salesman problem. In: Proc of the fourth international conference on computer and information technology. 2004. p. 796–800.
[215] Papert S. Mindstorms: children, computers, and powerful ideas. New York: Basic Books; 1980.
[216] Parekh R, Yang J, Honavar V. Constructive neural-network learning algorithms for pattern classification. IEEE Trans on Neural Networks 2000;11(2):436–51.
[217] Parijs LV, Abbas AK. Homeostasis and self-tolerance in the immune system: turning lymphocytes off. Science 1998;280:243–8.
[218] Paton R, editor. Computing with biological metaphors. Chapman & Hall; 1994.
[219] Pattee HH. Simulations, realizations, and theories of life. In: Langton CG, editor. Artificial life. Addison-Wesley; 1988. p. 63–77.
[220] Păun G, Cutkosky SD. Membrane computing. Springer; 2002.
[221] Păun G, Rozenberg G, Saloma A. DNA computing: new computing paradigms. Springer; 1998.
[222] Păun G, Rozenberg G, Salomaa A. DNA computing. Springer; 2006.
[223] Peitgen H-O, Jürgens H, Saupe D. Chaos and fractals: new frontiers of science. Springer; 1992.
[224] Perelson AS. Immune network theory. Imm Rev 1989;110:5–36.
[225] Perelson AS, Oster GF. Theoretical studies of clonal selection: minimal antibody repertoire size and reliability of self-nonself discrimination. J Theor Biol 1979;81:645–70.
[226] Perelson AS, Hightower R, Forrest S. Evolution and somatic learning in V-region genes. Research in Immunology 1996;147:202–8.
[227] Perlovsky L. Toward physics of the mind: concepts, emotions, consciousness, and symbols. Physics of Life Reviews 2006;3:23–55.
[228] Perrone MP, Cooper LN. When networks disagree: ensemble method for neural networks. In: Mammone RJ, editor. Artificial neural networks for speech and vision. New York: Chapman & Hall; 1993. p. 126–42.
[229] Pittenger AO. An introduction to quantum computing algorithms. Birkhäuser; 2000.
[230] Poli R, Chio CD, Langdon WB. Exploring extended particle swarms: a genetic programming approach. In: Proceedings of the 2005 conference on genetic and evolutionary computation. 2005. p. 169–76.
[231] Prusinkiewicz P, Lindenmayer A. The algorithmic beauty of plants. Springer; 1990.
[232] Ray TS. An evolutionary approach to synthetic biology. Artificial Life 1994;1(1/2):179–209.
[233] Ray TS, Hart J. Evolution of differentiated multi-threaded digital organisms. Artificial Life 1998;VI:295–304.
[234] Reif JH. The design of autonomous DNA nanomechanical devices: walking and rolling DNA. Lecture notes in computer science, vol. 2568. 2003. p. 22–37.
[235] Reif JH. Parallel molecular computation. In: Proc of the 7th ACM symposium on parallel algorithms and architecture. 1995. p. 213–23.
[236] Reif JH. Computing: successes and challenges. Science 2002;296(5567):478–9.
[237] Rennard J-P. Perspectives for strong artificial life. In: de Castro LN, Von Zuben FJ, editors. Recent developments in biologically inspired computing. Idea Group Inc; 2004. p. 301–18 [chapter 12].
[238] Resnick M. Turtles, termites, and traffic jams: explorations in massively parallel microworlds. MIT Press; 1994.
[239] Reynolds CW. Flocks, herds, and schools: a distributed behavioral model. Computer Graphics 1987;21(4):25–34.
[240] Reynolds CW. Steering behaviors for autonomous characters. In: Proc of the game developers conference; 1999 [on-line] http://www.red3d.com/cwr/papers/1999/gdc99steer.pdf.
[241] Room P, Hanan J, Prusinkiewicz P. Virtual plants: new perspectives for ecologists, pathologists and agricultural scientists. Trends in Plant Science 1996;1(1):33–8.
[242] Rosenblatt F. Principles of neurodynamics. Spartan Books; 1962.
[243] Roweis S, Winfree E, Burgoyne R, Chelyapov N, Goodman M, Rothemund P, Adleman L. A sticker based model for DNA computation. In: Baum E, Boneh D, Kaplan P, Lipton R, Reif J, Seeman N, editors. DNA based computers, Proc of the 2nd annual meeting. 1996. p. 1–27.
[244] Ruben AJ, Landweber LF. The past, present and future of molecular computing. Nature Reviews, Molecular Cell Biology 2000;1:69–72.
[245] Rumelhart DE, McClelland JL, The PDP Research Group. Parallel distributed processing: explorations in the microstructure of cognition, vol. 1. MIT Press; 1986.
[246] Schrödinger E. What is life?: With mind and matter and autobiographical sketches. Cambridge University Press; 1992.
[247] Schwefel H-P. Kybernetische Evolution als Strategie der Experimentellen Forschung in der Strömungstechnik, Diploma thesis, Technical University of Berlin, 1965 March.
[248] Secker A, Freitas A, Timmis J. A danger theory inspired approach to web mining. Lecture notes in computer science, vol. 2787. 2003. p. 156–67.
[249] Sekanina L. Evolvable components—from theory to hardware implementations. Springer; 2004.
[250] Selinger P. A brief survey of quantum programming languages. Lecture notes in computer science, vol. 2998. 2004. p. 1–6.
[251] Sengupta R, Dey N, Datta AK, Ghosh D. Assessment of musical quality of tanpura by fractal-dimensional analysis. Fractals 2005;13(3):245–52.
[252] Shi Y, Eberhart RC. Empirical study of particle swarm optimization. In: Proc of the IEEE congress on evolutionary computation. 1999. p. 1945–50.

[253] Shigenori N, Takamu GJ, Toshiki Y, Yoshikazu F. A hybrid particle swarm optimization for distribution state estimation. IEEE Trans on Power Systems 2003;18(1):60–8.
[254] Shmygelska A, Hoos HH. An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem. BMC Bioinformatics 2005;6(30):1–22.
[255] Shor PW. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing 1997;26(5):1484–509.
[256] Sienko T, Adamatzky A, Rambidi N. Molecular computing. MIT Press; 2003.
[257] Song CTP, Hall PS, Ghafouri-Shiraz H. Shorted fractal Sierpinski monopole antenna. IEEE Transactions on Antennas and Propagation 2004;52(10):2564–70.
[258] Steane AM. Introduction to quantum error correction. Proc Roy Soc Lond A 1996;452:2551–77.
[259] Steane A. Quantum computing. Reports on Progress in Physics 1998;61(2):117–73.
[260] Stepney S, Smith RE, Timmis J, Tyrrell AM. Towards a conceptual framework for artificial immune systems. Lecture notes in computer science, vol. 3239. 2004. p. 53–64.
[261] Stix G. Best kept secrets—quantum cryptography has marched from theory to laboratory to real products. Scientific American 2005;292(1):78–83.
[262] Stützle T, Hoos HH. MAX–MIN Ant system. Future Generat Comput Syst 2000;16(8):889–914.
[263] Su X, Smith LM. Demonstration of a universal surface DNA computer. Nucleic Acids Research 2004;32:3115–23.
[264] Sung HP, Yan H, Reif JH, LaBean TH, Finkelstein G. Electronic nanostructures templated on self-assembled DNA scaffolds. Nanotechnology 2004;15(10):S525–7.
[265] Sutton RS, Barto AG. Reinforcement learning: an introduction. Bradford Book; 1998.
[266] Szilard AL, Quinton RE. An interpretation for DOL systems by computer graphics. The Science Terrapin 1979;4:8–13.
[267] Timmis J, Bentley PJ, Hart E. In: Proceedings of the 2nd international conference on artificial immune systems. Lecture notes in computer science, vol. 2787. Springer; 2003.
[268] Timmis J, Neal M, Hunt J. An artificial immune system for data analysis. BioSystems 2000;55(1):143–50.
[269] Tizard IR. Immunology: an introduction. 4th ed. Saunders College Publishing; 1995.
[270] Trappenberg T. Fundamentals of computational neuroscience. Oxford University Press; 2002.
[271] Vapnik VN. The nature of statistical learning theory. Springer; 1995.
[272] Varela FJ, Coutinho A. Second generation immune networks. Imm Today 1991;12(5):159–66.
[273] Venkatraman S, Yen GG. A generic framework for constrained optimization using genetic algorithms. IEEE Trans Evolutionary Computation 2005;9(4):424–35.
[274] Villalobos-Arias M, Coello Coello CA, Hernández-Lerma O. Convergence analysis of a multiobjective artificial immune system algorithm. Lecture notes in computer science, vol. 3239. 2004. p. 226–35.
[275] Wang Z-O, Zhu T. An efficient learning algorithm for improving generalization performance of radial basis function neural networks. Neural Networks 2000;13(4–5):545–53.
[276] White T, Pagurek B. Towards multi-swarm problem solving in networks. In: Proc of the 3rd Int Conf on multi-agent systems (ICMAS'98). 1998. p. 333–40.
[277] Widrow B, Hoff Jr ME. Adaptive switching circuits. WESCON Convention Record, vol. 4. 1960. p. 96–104.
[278] Wilke CO, Adami C. The biology of digital organisms. Trends in Ecology and Evolution 2002;17:528–32.
[279] Wohlberg B, De Jager G. A review of the fractal image coding literature. IEEE Transactions on Image Processing 1999;8(12):1716–29.
[280] Wolfram S. Cellular automata and complexity. Perseus Books; 1994.
[281] Xia Y, Feng G. On convergence conditions of an extended projection neural network. Neural Computation 2005;17(3):515–25.
[282] Yang X, Chiocetti J, Papadopoulos D, Sussman L. Fractal antenna elements and arrays 5/99. 1999. p. 34–46, Applied Microwave & Wireless, vol. 11 No. 5.
[283] Yannakakis GN. AI in computer games: generating interesting interactive opponents by the use of evolutionary computation. PhD thesis. University of Edinburgh, College of Science and Engineering, School of Informatics; 2005.
[284] Yao X. Evolving artificial neural networks. Proceedings of the IEEE 1999;87(9):1423–47.
[285] Yao X, Xu Y. Recent advances in evolutionary computation. Journal of Computer Science & Technology 2006;21(1):1–18.
[286] Zebulum RS, Pacheco MAC, Vellasco MMBR. Evolutionary electronics: automatic design of electronic circuits and systems by genetic algorithms. CRC Press; 2001.
[287] Zhang L, Zhou C, Liu X, Ma Z, Ma M, Liang Y. Solving multi objective optimization problems using particle swarm optimization. In: Proceedings of IEEE congress on evolutionary computation 2003 (CEC 2003), Canbella, Australia. 2003. p. 2400–5.
[288] Zheng Y, Ma L, Zhang L, Qian J. On the convergence analysis and parameter selection in particle swarm optimization. In: Proceedings of international conference on machine learning and cybernetics 2003. 2003. p. 1802–07.
[289] Zhou ZH, Wu J, Tang W. Ensembling neural networks: many could be better than all. Artificial Intelligence 2002;137(1–2):239–63.
[290] Zietsch B, Elston GN. Fractal analysis of pyramidal cells in the visual cortex of the galago (Otolemur Garnetti): regional variation in dendritic branching patterns between visual areas. Fractals 2005;13(2):83–90.
[291] Zingel T. Formal models of DNA computing: a survey. Proc of the Est Ac of Sci Phys Math 2000;49(2):90–9.