Adaptation and Hybridization in Nature-Inspired Algorithms

Iztok Fister[1,⋆], Damjan Strnad[1], Xin-She Yang[2], and Iztok Fister Jr.[1]

[1] University of Maribor, Faculty of Electrical Engineering and Computer Science
Smetanova ul. 17, 2000 Maribor, Slovenia
{iztok.fister,damjan.strnad,iztok.fister1}@um.si
[2] School of Science and Technology, Middlesex University, London NW4 4BT, UK
x.yang@mdx.ac.uk

Abstract. The aim of this chapter is to familiarize readers with the basics of adaptation and hybridization in nature-inspired algorithms as necessary for understanding the main contents of this book. Adaptation is a metaphor for flexible autonomous systems that respond to external changing factors (mostly environmental) by adapting their well-established behavior. Adaptation emerges in practically all areas of human activities as well. Such adaptation mechanisms can be used as a general problem-solving approach, though it may suffer from a lack of problem-specific knowledge. To solve specific problems with additional improvements of possible performance, hybridization can be used in order to incorporate a problem-specific knowledge from a problem domain. In order to discuss relevant issues as general as possible, the classification of problems is identified at first. Additionally, we focus on the biological foundations of adaptation that constitute the basis for the formulation of nature-inspired algorithms. This book highlights three types of inspirations from nature: the human brain, Darwinian natural selection, and the behavior of social living insects (e.g., ants, bees, etc.) and animals (e.g., swarm of birds, shoals of fish, etc.), which influence the development of artificial neural networks. evolutionary algorithms, and swarm intelligence, respectively. The mentioned algorithms that can be placed under the umbrella of computational intelligence are described from the viewpoint of adaptation and hybridization so as to show that these mechanisms are simple to develop and yet very efficient. Finally, a brief review of recent developed applications is presented.

Keywords: Computational intelligence, evolutionary algorithms, swarm intelligence, artificial neural networks, adaptation, nature-inspired algorithms.

1 Introduction

The noun adaptation originates from the greek word *ad aptare* which means *to fit to*. This word emerged primarily in biology and was later widened to other

⋆ Corresponding author.

I. Fister and I. Fister Jr. (eds.), *Adaptation and Hybridization in Computational Intelligence*, Adaptation, Learning, and Optimization 18, DOI: 10.1007/978-3-319-14400-9_1

areas as well. It designates a collective name for problems arising within different areas, e.g., genetic, artificial intelligence, economics, game theory, etc., encompassing the optimization problems of different difficulties regarding *complexity* and *uncertainty* [41]. Complexity means how much effort must be incorporated in order to solve a specific problem. Uncertainty denotes the environment in which a problem arises and typically, changes over time. In general, real-world problems are embodied within environments which are typical dynamic, noisy and mostly unpredictable.

An adaptive system undergoes acting operators that affects its structure. That means, such systems adapt to the changing conditions of the environment by modifying the structure. In fact, each system prepares itself for changes using the so-called *adaptive plan*; i.e., the set of factors controlling these changes [41]. The adaptive plan determines how the structures are changed in order to best fit to the changing environment. Typically, the adaptive plans are realized by developing the *operators* that determine how the changes of structures are performed. There are several plans (operators) that can be used for adapting to the environment. Which of these is the best depends on a *performance measure* in which the estimation of a plan is based. Selecting the proper performance measure depends on the domain from which the specific problem arises. On the other hand, the performance measure estimates the quality of the modified structure.

Many natural, as well as artificial systems, arising within different domains are adaptive in nature. Some of these systems, by their structures and performance measures, are illustrated in Table 1.

In genetics, the structure of an adaptation is a chromosome that undergoes the actions by the operators of crossover, mutation and inversion. The quality of an individual is measured by its fitness. The fitter the individual, the more chances it has to survive. Artificial intelligence looks for a program tool that imitates the behavior of the human brain, which should be able to learn, while

Table 1. Domains and corresponding structures, operators, and performance measures

Domain	Structures	Operators	Performance Measure
genetic	chromosome	mutation, crossover, inversion	fitness
artificial intelligence	program cleavage	learning	error function
production	goods/services	production activities	utility
game theory	strategies	rules	payoff
supramolecular chemistry	supermolecules	recognition,transcription, transformation	amount of energy and information
memetic computation	memes	transmission, selection, replication, variation	payoff

its performance is normally measured by the error function. The smaller the value of error function, the better the program is adapted to its environment.

Production is a process of combining various material and immaterial inputs (plans, know-how) in order to make something for consumption (the output). It is the act of creating outputs, goods or services which have values and contribute to the utility of individuals [35]. The higher the utility, the more the production process is optimized.

In a game theory, a game is a mathematical model of a situation of interactive decision making, in which every decision maker (or player) strives to attain his "the best possible outcome" [42]. Indeed, each player plays a move according to the strategy that maximize its payoff. The payoff matrix provides a quantitative representation of players' preference relations over the possible outcomes of the game. The strategy for player A is the winning strategy if for every move of player B, player A is the winner. A combination of moves must obey the game rules by all game players.

Supramolecular chemistry may be defined as "chemistry beyond the molecule", where two molecules (i.e., receptor and substrate) are assembled into supramolecules using intermolecular bonds [43]. Supramolecules undergo the actions such as molecular recognition, transformation, and translocation that may lead to the development of molecular and supramolecular species and can provide very complex functions. These species are capable of self-organizing, self-assembling and replicating by using molecular information. Here, the amount of energy and information is employed as the performance measure.

In memetic computation (MC), a meme represents a building block of information obtained by autonomic software agents obtained either by learning or by interacting with the surrounding agents which acts within a complex dynamic environment [24]. Indeed, memes can represent the agent's ideas and knowledge captured as memory items and abstractions (e.g., perceptions, beliefs, minds) [29]. The primary memetic operator is imitation [61], which takes place when the memes are transmitted, replicated or modified. These intelligent agents are also confronted by selection, where the agents with the higher payoffs in the previous generations have more chances for survival.

Although adaptation has emerged within different domains of human activities, it shares the similar characteristics, e.g., each adapted system has its structure on which operators are applied according to an adaptive plan, while the modified structure is estimated using a suitable performance measure. The higher the performance measure, the better the system adapts to its environment. As a result, only the best adapted structures can continue to develop and improve their characteristics. The less adapted ones are condemned to disappear. In this sense, adaptation can also be viewed as an optimization process.

Obviously, most real-world problems are hard to solve. This means that problems cannot be solved exactly by an algorithm enumerating all the possible solutions. They are too complex in terms of both the time and space necessary for obtaining solutions [40]. Therefore, these problems are usually solved approximately by using heuristic methods that guess the solution of the problem

in some (ideally smart) way. Although such a solution is not exact, it is good enough to be used in a practice.

Nowadays, algorithm developers often try to imitate the operations of natural processes by attempting to solve the harder, real-world problems. From the algorithm development point of view, there are three types of inspiration sources from nature:

- human brain,
- natural selection,
- behavior of some social living insects and animals.

The first source of inspiration has led to the emergence of the artificial intelligence, where the algorithm tries to mimic the operations of human brains in order to solve problems, where the main example is the artificial neural networks (ANNs) [39]. The second source of inspiration has led to the foundations of evolutionary algorithms (EA) [36] using the Darwinian natural selection [37], where the fittest individual in a population can survive during the struggle for existence. The third source of inspiration has closely related to the development of swarm intelligence (SI) [1,173] that mimics the social behavior of some living insects and animals [38]. Although such systems tend to obey simple rules, simple creatures such as ants are capable of performing autonomous actions, they are still capable of doing great things, e.g., building magnificent anthills, when acting together within a group. All three mentioned nature-inspired algorithms can be placed under the umbrella of computational intelligence (CI). The algorithms belonging to this family share the same characteristics, i.e., they are capable of solving the problems on some sophisticated, intelligent way.

On the other hand, the behavior of an optimization algorithm is controlled by its parameters (also strategy or control parameters). These parameters mostly stay fixed during the algorithm's run. However, this is in contrast to the real-world, where the good starting values of parameters could become bad during the run. As a result, a need has been emerged to modify them during the run. Here, the adaptation of control parameters can be used as well, where the values of the control parameters are modified during the run in order to best suit the demands of the search process.

In addition, many traditional algorithms, especially gradient-based methods, exist that contain a lot of domain-specific knowledge within algorithm structures. Contrary, the general problem solver methods, especially nature-inspired population-based algorithms like EAs and SI, are capable to obtain the moderate results on all classes of optimization problems. In order to connect the general problem solver methods with the traditional heuristics, the hybridization of nature-inspired population-based algorithms with traditional heuristic algorithms has been performed. Such hybridized algorithms incorporate a problem-specific knowledge into algorithms' structures and are therefore more suitable for solving the specific problems. Using more problem-specific knowledge, these algorithms may overcome limitation imposed by the No-Free Lunch theorem [18] stating that two algorithms are equivalent when comparing across all classes of problems. According to Chen et al. [24], the hybridized algorithms evolved over

simple hybrids, via adaptive hybrids to memetic automation. Simple hybrids often represent a hybridization of population-based CI algorithms with local search heuristics. The result of connecting the adaptation with hybridization has led to adaptive hybrids. The last step in the integration of adaptation with hybridization forms a part of memetic computing, where, in addition to the parameters, other algorithmic structures can also be adapted.

The remainder of this chapter is organized as follows. Section 2 deals with optimization problems and their complexity. The origin of adaptation within natural systems is the subject of Section 3. Section 4 analyzes the nature-inspired algorithms. In line with this, the ANN, EA and SI-based algorithms are taken into account. Section 5 highlights key characteristics of adaptation and diversity in CI. Section 6 deals with a description of hybridization methods in CI. A brief review of recent application arisen in CI is given in Section 7. Finally, some conclusions are drawn in Section 8.

2 Classification of Problems

From a system analysis point of view, problem-solving can be seen as a system consisted of three components: input, output, and model (Fig. 1). The model transforms input data to output data. If the model is known, the output data can be determined by each set of input data. The problem can also be placed differently, i.e., which input data produces specific output data by a known model. Finally, knowing the input and output data, the problem is how to find a model that transforms the specific input data to the output data.

Fig. 1. Problems and System Analysis

In line with this, three classes of problems can be defined with regard to one of the unknown components within system analysis, as follows:

- optimization: the input data that satisfies a criterion of optimality are searched for by a known model and known output data,
- simulation: a set of known input data are applied to the known model in order to simulate the output data,
- modeling: searching for a (mathematical) model is performed, which can transform the known input data to the known output data, at a glance.

The optimization and simulation/modeling problems are described in the next subsections in more detail.

2.1 Optimization Problems and Their Complexity

When solving optimization problems, the output value needs to be determined with a set of input data, a model for transforming the input data into output, and a goal prescribing the optimal solutions. Optimal solutions are feasible solutions the values of which are either minimal or maximal. These values can be written as $y^* = f(\mathbf{y}^*)$, while their optimal values as $f^*(\mathbf{y})$. Only one set of input data can be set on the input. This set is therefore known under the name instance. The set of all instances that can appear on the input constitute an optimization problem P. Formally, the optimization problem is defined as quadruple $P = \langle I, S, f, goal \rangle$, where

- I is a set of instances of problem P,
- S is a function assigning each instance $\mathbf{x} \in I$ to a set of feasible solutions $S(\mathbf{x})$, where $\mathbf{x} = \{x_i\}$ for $i = 1 \ldots n$ and n determines a dimensionality of the problem P,
- f is an objective function assigning a value $f(\mathbf{y}) \in \mathbb{R}$ to each feasible solution $\mathbf{y} \in S(\mathbf{x})$,
- the *goal* determines whether the feasible solution with the minimum or maximum values is necessary to search for.

In computational intelligence, the fitness function is employed in place of the objective function because using the equality $\min(f(\mathbf{y})) = \max(-f(\mathbf{y}))$ the maximal values of objective function can be transformed into searching for the minimal values of the fitness function.

The optimization problems may be emerged within one of three possible forms, as follows:

- *constructed* form, where the optimal values of variables $\mathbf{y}^*$ and the corresponding value of objective function $f^*(\mathbf{y})$ needs to search for a given instance $\mathbf{y} = S(\mathbf{x})$,
- *non-constructed* form, where the optimal value of objective function $f^*(\mathbf{y})$ needs to search for a given instance $\mathbf{y} = S(\mathbf{x})$,
- *decision* form, where the problem is to identify whether the optimal value of the objective function is better than some prescribed constant K, i.e., either $f^*(\mathbf{y}) \leq K$, when $goal = \min$ or $f^*(\mathbf{y}) \geq K$, when $goal = \max$.

Optimization problems can be divided into three categories, i.e., problems using: the *continuous* variables, the *discrete* (also *combinatorial*) variables, and the *mixed* variables. The first category of problems searches for the optimum value in an infinite set of real numbers $\mathbb{R}$. Variables are taken from a finite set by discrete problems, while they may be either discrete or continuous by the mixed problems.

In general, algorithms are procedures for solving problems according to certain prescribed steps [2]. Usually, these procedures are written in some programming language. If a certain algorithm solves all instances I of the specific problem P then it can be said that the algorithm solves this problem completely. Here, the algorithm which solves this problem the most efficiently is typically searched

for. The efficiency of algorithms is normally estimated according to the time and space occupied by the algorithm during a run. Generally, the more efficient algorithms are those that solve problems the fastest.

Time complexity is not measured by the real-time as required for solving the problem on a concrete computer because this measure would not be fair. Algorithms can be run on different hardware or even on different operating systems. In general, the problem or instance size is therefore measured in some informal way which is independent of the platform on which the algorithm runs. Therefore, time complexity is expressed as a relation that determines how the time complexity increases with the increasing problem size. Here, we are not interested in the problem size, but in how the instance size influences on the time complexity. If the algorithm solves a problem of size n, for example, with a time complexity $C \cdot n^2$ for some constant C means that the time complexity of the algorithm is $O(n^2)$ (read: of order n^2). The function $O(n^2)$ determines an asymptotic time complexity of the algorithm and limits its upper bound.

If the time complexity of the algorithm is exponential, i.e., $O(2^n)$, it can be argued that the problem is hard. As a result, these kinds of problems belong to a class of nondeterministic-polynomial hard problems (i.e., NP-hard) [40]. Classical combinatorial problems like the Traveling Salesman Problem (TSP) [44], the Graph Coloring Problem (GCP) [45], etc. are members of this class.

2.2 Simulation/Modeling Problems

The behavior of real-world facilities or processes (also systems) can be described in the form of mathematical or logical relationships. In general, these real world systems are too complex for expressing their behavior with exact mathematical methods. Therefore, analytical solutions of this system's behavior are not possible. As a result, the system is studied by simulation, where the mathematical model of the system is built on a digital computer. The task of simulation is to evaluate a model numerically by known input variables in order to obtain output variables matching the expected real world values as closely as possible.

In this chapter, modeling problems (in the narrow sense) refer to supervised learning, where on the basis of observing some examples of input-output pairs, the system learns a model that maps input data to output data. Supervised learning can be defined formally as follows. Let a training set be given with N instances of input-output pairs in the form $(\mathbf{x}_1, \mathbf{y}_1), \cdots, (\mathbf{x}_N, \mathbf{y}_N)$, where each $\mathbf{y}_i$ is generated by an unknown function $\mathbf{y} = f(\mathbf{x})$. The task is to discover a function h that approximates the true function f [39].

The function h represents a hypothesis that is validated throughout all input-output pairs during the learning process. The learning process is finished when the search space of all possible hypotheses is searched for and none of these are rejected. Moreover, the learned model h must also perform well on the so-called test set of input-output pairs that are distinct from the training set.

When the elements of output vector $\mathbf{y}$ belong to a finite set of values, such a learning problem becomes a classification problem. On the other hand, when these elements are real values, the learning problem is also known as regression.

3 Biological Foundations of Natural Adaptation

In natural evolution, adaptation indicates a genetic as well as non-genetic modification of individuals during more generations. Moreover, this term is usually used as being a synonymous for measure of fitness, i.e., a characteristic that increases during generations. What does an individual adapt to? More frequently, here goes about adapting to conditions of environment or ecological niche, i.e., an area that is occupied by individuals living in a particular community because of common exploitation of resources in the environment [46]. Too specific adaptation of a particular ecological niche can lead to speciation [37].

Darwin's finches (also Galápagos finches) are one of the most famous examples of speciation using adaptation, where a group of about fifteen finch species with common ancestors occupied specific ecological niches and adapted to different food sources with different body sizes and beak shapes. Indeed, only the best adapted individuals survived. The process of so-called adaptive radiation [3], in which individuals diversify rapidly into a multitude of new forms, had been started when finch ancestors originated from South America occupied an island in Galápagos archipelago closest to the continent.

The adaptive radiation as an origin of evolutionary diversity opens up the question as to when and why comes to the speciation. Darwin in 1859 [37] answered with an allopatric model of speciation whereby the evolutionary diversity was caused by geographical separation of the population.

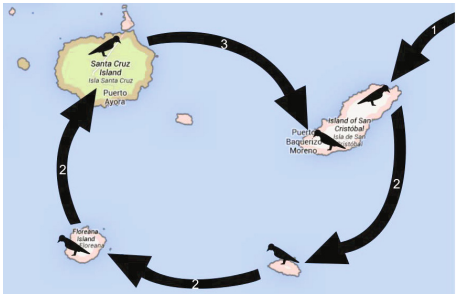


Fig. 2. Galápagos archipelago

Speciation and formation of new Galápagos finches were carried over three phases (Fig. 2):

- The population of finches colonized an island closest to the continent. This population underwent the rules of natural selection.
- Part of the population separated from the group and colonized the next island. They adapted themselves to new environmental conditions, because a distribution of food sources on the next island was different. As a result, only the most adapted to the new conditions with the body size and the shape of

their beaks could survive. Additionally, geographically separated populations underwent changes of their reproduction materials through mutation.
– The process of colonizing the other islands of the Galápagos archipelago repeated until finally, the conquering population recolonized the site island from which the adaptive radiation started. As a result, the new population meets its ancestor population.

The meeting of these two populations may have caused the individuals of both populations:

– to mate among themselves and the offsprings became more successful than their parents,
– to mate among themselves and the offsprings became less successful than their parents,
– not to mate among themselves.

In the first case, both populations merge together into a single one, while in the third case the individuals of both populations are so different that the mating was impossible. In this worst case, reproduction isolation happens and prevents mating between the individuals of two different populations. However, the most interesting is the second case that represents a process of adaptive radiation that could cause population isolation over a longer period of time. However, this isolation is just a precondition for speciation.

More recently views on the adaptive radiation and speciation of Darwin's finches have cast doubt in the correctness of the allopatric model [37]. Indeed, it seems that the proximity of the Galápagos islands might prevent the existence of geographical isolation and therefore, the finches could freely travel between islands. This fact also suggests that more populations need to live on the same place at the same time.

Today, a sympatric model has been established that argues speciation without geographical isolation [3]. In this model, new species appear as a result of adaptation to ecological niches. When individuals of a sympatric species mate between themselves, then the fitness of their offspring usually decreases. The natural selection quickly eliminates such individuals from the population. On the other hand, differences in the reproductive materials changed by mutations can also cause a reproduction barrier whereby individuals of different populations do not mate between themselves and thus speciation can occur.

Differences in reproduction material represent a reproduction barrier when mating has been performed. Usually, the reproduction barrier can emerge before the mating takes place. Interestingly, each male finch uses a similar kind of courtship. Thus, it is not important how males appear, but how they look. Usually, males differ between themselves according to the size and the shape of their beaks rather than the birds' plumages. As a result, the size and the shape of the beaks adapted to the local food sources can cause a reproduction barrier between individuals of sympatric populations.

Furthermore, the reproduction isolation can also be caused by differences in the acquired characteristics of individuals (i.e., ecological isolation), e.g., sounds

that have been learned by males from their parents and which are susceptible by females of the same population. The sound is independent of the reproduction material, although morphological characteristics of individuals are written in genes (e.g., the size and the shape of beaks) can have an impact on the volume and pitch of sound articulated by the bird.

Interestingly, Wright's concept of adaptive landscape [4] can be used to illustrate the morphological characteristics of Darwin's finches according to various food sources on the Galápagos islands. Both different morphological characteristics, i.e., the body size and the shape of beaks, are represented as two coordinate axes in a 3-dimensional coordinate system, while the third axis represents selective advantages or disadvantages of morphological characteristics of a specific individual in regard to the food sources.

The adaptive landscape of morphological characteristics versus body sizes and beak shapes can change over the longer period of time. Therefore, such landscape is also named *dynamic adaptive landscape*. Similarly to the conditions in the environment have changed over time, also the heights and positions of hills are changed in the adaptive landscape. For instance, the height of the hill is lowered, a valley between two hills is increased or two hills move closer to each other or move away from each other. Various populations of Darwin's finches adapt to these changes in the environment. If, for example, two hills are moved closer to each other because of frequent earthquakes on Galápagos archipelago, two or more populations of Darwin's finches come together, while if the hills are moved away the groups of finches are being separated. Speciation appears when the specific population colonizes the peak of a hill. Each hill is occupied by exactly one finch population with the body size and the shape of beaks adapted to the specific food source. As a result, fifteen ecological niches can be discovered on the Galápagos archipelago, on which exactly the same number of finch species have appeared.

In computational intelligence, the adaptive landscape is known as the *fitness landscape*. Furthermore, the speciation is more frequently used by solving multimodal problems, where more equivalent problem solutions (i.e., more peaks within the fitness landscape) are maintained during the algorithm run. In fact, each peak represents an ecological niche appropriate for speciation [47].

Therefore, different landscapes (from different problems) may pose different challenges to different algorithms. It is not possible in general to adapt to all landscapes at the same time. As a result, different algorithms may perform differently for different problems. In order to solve this a broad spectrum of various problems, developers of new algorithms draw inspirations from different natural systems. Nature-inspired algorithms are the most generalized terms and we will discuss nature-inspired algorithms in greater detail in the next section.

4 Nature-Inspired Algorithms

Nature-inspired algorithms are very diverse. Loosely speaking, we can put nature-inspired algorithms into three categories: artificial neural networks, evolutionary

algorithms and swarm intelligence. It is worth pointing out that such categorization here is not rigorous. However, it is mainly for the convenience of discussions in this chapter.

4.1 Algorithm as an Iterative Process

Mathematical speaking, an algorithm A is an iterative process, which aims to generate a new and better solution $\mathbf{x}^{(t+1)}$ to a given problem from the current solution $\mathbf{x}^{(t)}$ at iteration or (pseudo)time t. It can be written as

$$\mathbf{x}^{(t+1)} = A(\mathbf{x}^{(t)}, p), \tag{1}$$

where p is an algorithm-dependent parameter. For example, the Newton-Raphson method to find the optimal value of $f(\mathbf{x})$ is equivalent to finding the critical points or roots of $f'(\mathbf{x}^{(t)}) = 0$ in a d-dimensional space. That is,

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \frac{f'(\mathbf{x}^{(t)})}{f''(\mathbf{x}^{(t)})} = A(\mathbf{x}^{(t)}). \tag{2}$$

Obviously, the convergence rate may become very slow near the optimal point where $f'(\mathbf{x}) \to 0$. Sometimes, the true convergence rate may not be as quick as it should be. A simple way to improve the convergence is to modify the above formula slightly by introducing a parameter p as follows:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - p\frac{f'(\mathbf{x}^{(t)})}{f''(\mathbf{x}^{(t)})}, \quad p = \frac{1}{1 - A'(\mathbf{x}^*)}. \tag{3}$$

Here, $\mathbf{x}^*$ is the optimal solution, or a fixed point of the iterative formula.

The above formula is mainly valid for a trajectory-based, single agent system. For population-based algorithms with a swarm of n solutions $(\boldsymbol{x}_1^{(t)}, \boldsymbol{x}_2^{(t)}, ..., \boldsymbol{x}_n^{(t)})$, we can extend the above iterative formula to a more general form

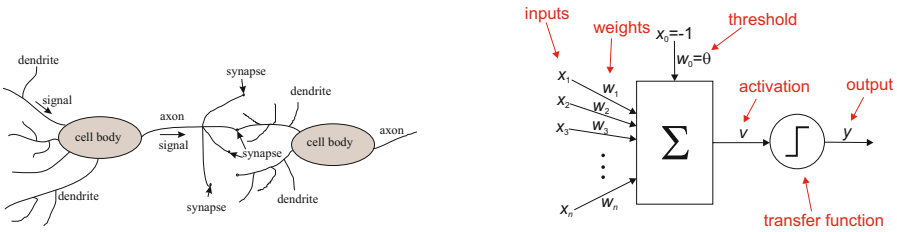
$$\begin{pmatrix} \boldsymbol{x}_1 \\ x_2 \\ \vdots \\ \boldsymbol{x}_n \end{pmatrix}^{(t+1)} = A\Big((\boldsymbol{x}_1^{(t)}, \boldsymbol{x}_2^{(t)}, ..., \boldsymbol{x}_n^{(t)}); (p_1, p_2, ..., p_k); (\epsilon_1, \epsilon_2, ..., \epsilon_m)\Big) \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_n \end{pmatrix}^{(t)}, \tag{4}$$

where $p_1, ..., p_k$ are k algorithm-dependent parameters and $\epsilon_1, ..., \epsilon_m$ are m random variables. An algorithm can be viewed as a dynamical system, Markov chains and iterative maps [173], and it can also be viewed as a self-organized system [174].

4.2 Artificial Neural Networks

The human brain consists of a network of interconnected neural cells (also-called neurons) which communicate using electrochemical signaling mechanisms. The

main part of a human neuron (Fig. 3.a) is the cell body that contains a cell nucleus [39]. The cell body branches out with a number of fibers (dendrites) and a single long fiber named an axon. The neuron accepts the incoming signals from its neighbors' axons through dendrite tips at junctions called synapses, which inhibit or amplify the signal strength. After the processing of accumulated inputs inside the nucleus the output signal is propagated through the axon to neurons down the communication line. The brain function is evolved through short-term and long-term changes in the connectivity of the neurons, which is considered as learning.



(a) Biological neuron (b) Artificial neuron

Fig. 3. Human and artificial neuron

There is a natural desire to compare the performance of the human brain with the performance of a digital computer. Like the brain, today's computers are capable of highly parallel processing of signals and data. Interestingly, today's capacity of a digital computer is comparable to the capacity of the human brain. Moreover, they are capable of parallel processing. On the other hand, human brain do not use all of their neurons simultaneously. If it is further assumed that according to Moore's law [14], the memory capacity of digital computers doubles approximately every two years, if this trend continues, it is obviously possible that the singularity point [15] at which the performance of digital computers will be greater than those of the human brain has to be reached. Although computer intelligence has virtually unlimited capacity, this does not mean that truly intelligence will emerge automatically. It is still a challenging, unresolved task to figure out how to use such resources to produce any useful intelligence.

The *Artificial neural network* (ANN) is a simplified and inherently adaptive mathematical model of the human brain. The elementary part of every ANN is the *artificial neuron* (Fig 3.b), which is modeled after the biological brain cell. In an ANN the neurons communicate through weighted connections that simulate the electrochemical transfer of signals in the brain. Many different ANN topologies and neuron models have been presented in the past, each developed for a specific type of machine learning task like classification, regression (i.e., function approximation), or clustering. By far the most practically employed type of ANN is the multi-layered feed-forward neural network that consists of the McCulloch-Pitts type of artificial neuron [16].

The structure of a classical feedforward multi-layered neural network, commonly known as a *multi-layer perceptron* (MLP), is shown in Figure 4. The external input signals $x_i, 1 \leq i \leq n$, enter the network on the left and flow through multiple layers of neurons towards the outputs $o_i, 1 \leq i \leq m$, on the right. The neuron connectivity exists only from the previous layer to the next one, so the outputs of neurons in layer $l-1$ serve as inputs to the neurons of layer l. There is no interconnection of neurons within the layer, no backward connections, and no connections that bypass layers. In a MLP network with L layers, the first $L-1$ are called *hidden layers* and the last one is called the *output layer*. Two hidden layers are enough for most practical purposes. We shall use h_i to denote the number of neurons in the i-th hidden layer and m to denote the number of neurons in the output layer (i.e., the number of network outputs). We will use the compact notation $n/h_1/h_2/\ldots/h_{L-1}/m$ to describe such MLP network with n external inputs.

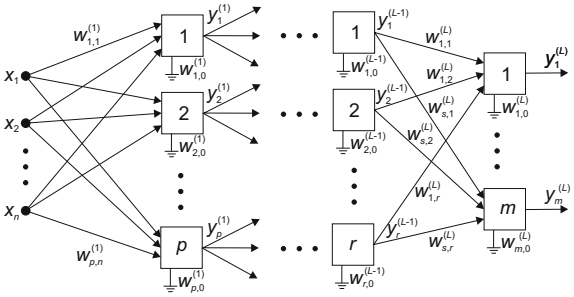


Fig. 4. Multi-layer feed-forward neural network

Every connection within the MLP network is assigned a real-valued weight that amplifies or inhibits the signal traveling over the connection. We will use notation $w_{ij}^{(l)}$ to denote the weight on the j-th input to the i-th neuron in layer l. The function of a MLP network with fixed structure is determined by a set of weights on all of its connections.

Neurons in a MLP function as simple processors that gather the weighted signals on their input lines and transform them into a single numerical output. In the McCulloch-Pitts neuron model shown in Fig. 3.b this is performed in two steps. The summation unit adds weighted inputs and shifts the result by an additional intercept parameter θ called *threshold* or *bias* to produce the neuron *activation* value v in the following way:

$$v = \sum_{i=0}^{n} w_i x_i, \tag{5}$$

where $\mathbf{x} = \{x_0, \ldots, x_n\}$ is the augmented input vector with $x_0 = -1$ and $\mathbf{w} = \{w_0, \ldots, w_n\}$ is the corresponding augmented weight vector with $w_0 = \theta$.

In the second step the activation value is injected into the *transfer function* ϕ to obtain the neuron output y:

$$\mathbf{y} = \phi(v). \tag{6}$$

The Heaviside step function is used in place of ϕ for classification tasks, while for regression tasks the popular choice for ϕ is the logistic function σ:

$$\sigma(v) = \frac{1}{e^{-v/\rho}}. \tag{7}$$

Here, ρ is the sigmoid slope parameter with default value 1. Fig. 5 shows the step function on the left and the logistic function for various values of ρ on the right-hand side. When a signed version of the sigmoid transfer function is required, the common choice is the hyperbolic tangent.

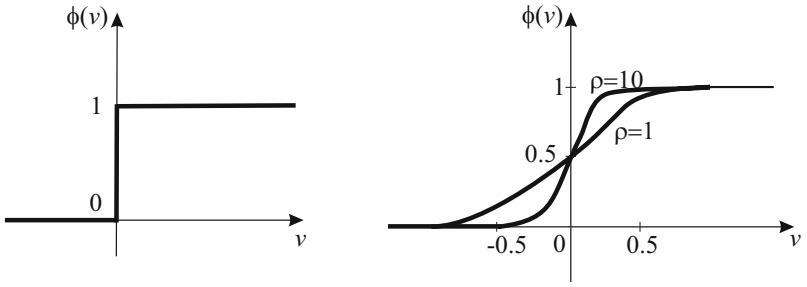


Fig. 5. The step (left) and the sigmoid (right) activation function

The flow of signals in a MLP network with structure $n/h_1/h_2/\ldots/h_{L-1}/m$ can be described in a unified form as:

$$y_i^{(l)} = \phi\left(\sum_{j=0}^{h_{l-1}} w_{ij}^{(l)} y_j^{(l-1)}\right), \quad 1 \leq i \leq h_i; 1 \leq l \leq L \tag{8}$$

where $h_0 = n$, $y_i^{(0)} = x_i$, $h_L = m$, and $o_i = y_i^{(L)}$.

Weights represent the programmable part of neural network. In order to perform a specific task, we need to train the MLP, i.e., adjust the weights using a set of training samples with known input-output mappings. This is an example of *supervised learning*, which is used in classification tasks with existing records of correctly labeled patterns or regression tasks with known values of an unknown nonlinear map in a given set of points.

The weight adaptation in neural networks is achieved by iterative training algorithms, in which the input parts of the training samples are presented to the network in succession. A cycle in which all of the training samples are introduced on the network input is called an *epoch*. The better known supervised training

method for the MLP is the *error back-propagation* algorithm. For each presented input, the computed network output $\mathbf{o}$ is compared with the target vector $\mathbf{d}$ to obtain the prediction error. The usual error measure E in back-propagation training is the *mean squared error* (MSE) of the output neurons:

$$E = \frac{1}{m}(\mathbf{d} - \mathbf{o})^T(\mathbf{d} - \mathbf{o}) \tag{9}$$

The weights are then updated in the direction of the negative gradient $\partial E/\partial \mathbf{w}$ to reduce the error in the next iteration.

Training continues until the maximum number of epochs is reached or the average MSE error for the epoch falls below some prescribed tolerance ϵ. General methods like cross-validation to prevent over-fitting can also be used for premature training termination. The complete back-propagation training algorithm is summarized in Algorithm 1.

Algorithm 1. Pseudo-code of back-propagation ANN

```
repeat
    initialize weights
    for all examples(x,y) do
        propagate the inputs forward to obtain the outputs
        propagate deltas backwards from output layer to input layer
        update every weight in network with deltas
    end for
until termination criteria met
return artificial neural network
```

Training continues until the maximum number of epochs is reached or the average MSE error for the epoch falls below some prescribed tolerance ϵ. General methods like cross-validation to prevent over-fitting can also be used for premature training termination. The complete back-propagation training algorithm is summarized in Algorithm 1.

4.3 Evolutionary Algorithms

EAs found their origins for basic operations from the Darwinian evolutionary theory of the survival of the fittest [37], where the fitter individuals in nature have more chances to survive in the struggle for survivor. Thus, the fitter individuals are able to adapt better to changing conditions of the environment. The lesser fit individuals are gradually eliminated from the population by natural selection.

Darwinian theory of survival of the fittest refers to a macroscopic view of natural evolution [36]. Today, it is known that all characteristic traits that define the behavior of an individual are written in genes as fundamental carriers of heredity. Individuals' outer characteristics (also phenotype) are determined in genes (also genotype). The view on these individuals as inner characteristics

is also known as the microscopic view of natural evolution. As matter of fact, the phenotypic characteristics are encoded into genotypes. Unfortunately, this encoding is not one-to-one, i.e., the genotype-phenotype mapping is not injective because one phenotype trait can be determined by more genes. On the other hand, a genetic material is passed onto the new generation using the process of reproduction. Reproduction consists of two phases: crossover and mutation. In the former phase, the genetic material from two parents are combined in order to generate offspring with new traits, while in the latter phase, the genetic material of the offspring may be randomly modified.

In order to introduce this Darwinian natural evolution in EAs, some links between the concepts of both domains should be performed [36]. Natural evolution is handled by a population of individuals living in an environment that changes over the time (also dynamic). On the other hand, EAs use the population of candidate solutions. The environment can be taken as the problem space. Similarly, the natural reproduction process is simulated by operators of crossover and mutation in EAs. Finally, the fitness of the individual in natural evolution represents the quality of the candidate solution in EAs. A pseudo-code of EA is presented in Algorithm 2, where two selection operators are supported in EAs. In the first selection (function *select parents*), two parents are selected for crossover, while in the second (function *select candidate solution for the next generation*), the candidate solutions are determined for the next generation. When the *generational* model of population is selected, the whole population is replaced in each generation, while using the *steady-state* model only the worst part of the population is replaced by the best offsprings.

Algorithm 2. Pseudo-code of evolutionary algorithm

```
initialize population with random candidate solutions
evaluate each candidate solution
while termination criteria not met do
    select parents
    recombine pairs of parents
    mutate the resulting offspring
    evaluate each candidate solution
    select candidate solution for the next generation
end while
return best agent
```

Evolutionary computation (EC) was inspired by Darwinian theory of natural evolution. EC is a contemporary term that captures all the algorithms arising from the principle of natural selection. Consequently, all algorithms that have been emerged within this EC domain are known under the name EAs. Loosely speaking, EAs can be divided into the following types (Fig. 6):

- Genetic Algorithms (GA) [47],
- Genetic Programming (GP) [49],

– Evolution Strategies (ES)[46],
– Evolutionary Programming (EP) [48],
– Differential Evolution (DE) [13].

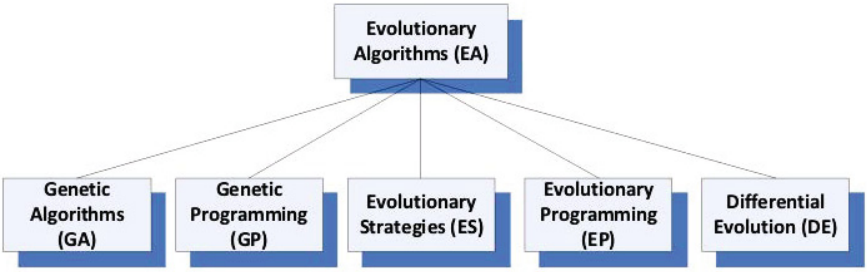


Fig. 6. Primarily, EAs differ from each other in terms of the representation of solutions. For example, GAs operate with a population of mainly binary represented solutions, ESs use real-valued elements of solutions, GPs represent solutions as trees implemented in Lisp programming language, while EPs employ the solutions represented as finite state automata.

EAs have been successfully applied to different areas of optimization, modeling and simulation, where problems cannot be solved sufficiently using traditional methods such as gradient-based methods.

4.4 Swarm Intelligence

Swarm intelligence concerns the studies of the collective behavior of multi-agent and decentralized systems, which may be self-organized and evolving. This term was probably first used by Beni in 1989 [1], when he developed cellular robots consisted of simple agents communicating by interactions with other agents within the neighborhood.

In nature, some social living insects (e.g., ants, bees, termites, etc.) and animals (e.g., flocks of birds, schools of fishes, etc.) may show some characteristics that may be classified as swarm intelligence (Fig. 7). Though individual agents such as ants and bees may follow simple rules, they can carry out complex tasks collectively. In other words, their decision making is decentralized, while they are self-organized and act consistently with the intentions of the group. Such interactions between individuals (such as particles) are local and rule based.

Interactions between particles in a swarm can be direct or indirect. In the indirect case, two particles are not in physical contact with each other because a

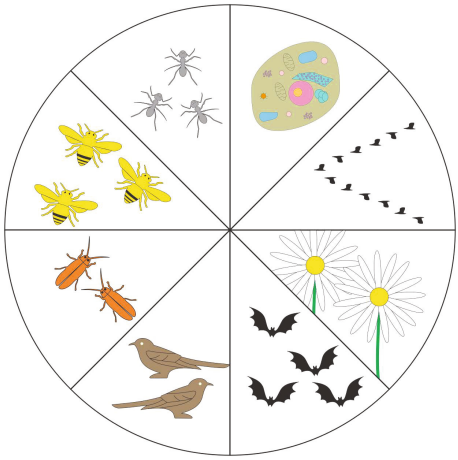


Fig. 7. Nature-inspired SI-based algorithms - The picture presents the sources of inspiration from nature for developing the following SI-based algorithms that follow in the clockwise direction: natural immune systems, particle swarm optimization, flower pollination algorithm, bat algorithm (echolocation), cuckoo search (to lay own eggs into other birds' nests), fireflies (bioluminescence), bee (foraging of nectar) and ant colonies(pheromone)

communication is performed via modulation of the environment [38]. For example, ants deposit pheromones on their way back from a profitable food source and other ants will follow paths marked with pheromone. In that way information is simply spit out without controlling who receives it. In the direct case, information is transferred directly without modulation of environment. A good example of such an interaction mechanism is the honeybees' 'waggle dance' to encode the spatial information: the direction and the distance to the nectar source. The quality of a new food source is assessed by the forager gauge, based on the sugar content of the nectar, the distance from the colony and the difficulty with which the nectar can be collected.

SI-based algorithms are population-based, which uses multiple interacting agents or particles. Each particle has a position and velocity where the position usually represents a solution to the problem of interest. Their interaction may be described by some mathematical equations, based on the idealized characteristic for the collective behavior of imitated insects or animals (e.g., swarm of birds, fireflies, etc.). In most SI-based algorithms, all solutions are moved towards the best candidate solution and thus, the new better solutions can be obtained. Sometimes, problems arise when the best solution cannot be improved anymore. In this case, *stagnation* emerges. However, stagnation may be avoided using an additional mechanisms like local search heuristics, though there is no guarantee that it will solve the stagnation issue. The pseudo-code of the generic SI-based algorithm is shown in Algorithm 3.

Algorithm 3. Pseudo-code of swarm intelligence algorithm

```
initialize swarm within bounds
evaluate all particles
while termination criteria not met do
    move all particles
    evaluate all particles
    find the best particle
end while
return best particle
```

The main characteristics of SI-based algorithms are as follows [38]:

- decentralization via rule-based models,
- interaction among particles is carried locally (collective behavior),
- particle behavior is subordinated to the system behavior (self-organization),
- adapting to changes in the landscape (reasonable robust and flexible).

Some representative SI-based algorithms are as follows:

- Artificial Immune Systems (AIS) [5],
- Particle Swarm Optimization (PSO) [8],
- Flower Pollination (FPA) [11],
- Bat Algorithm (BA) [9],
- Cuckoo Search (CS) [12],
- Firefly Algorithm (FA) [10],
- Artificial Bee Colony (ABC) [7],
- Ant Colony Optimization (ACO) [6].

It is worth pointing out that we can only cover and discuss less than 10% of all different SI-based algorithms in this brief review. However, the development of new types of the SI-based algorithms is not finished yet. Almost every day new SI-based algorithms have been emerging. In this way, there is no doubt that this area will become more active in the near future.

5 Adaptation and Diversity in Computational Intelligence

Adaptation in nature-inspired algorithms can take many forms. For example, the ways to balance exploration and exploitation are the key form of adaptation [175]. As diversity can be intrinsically linked with adaptation, it is better not to discuss these two features separately. If exploitation is strong, the search process will use problem-specific information (or landscape-specific information) obtained during the iterative process to guide the new search moves; this may lead to the focused search and thus reduce the diversity of the population, which

may help to speed up the convergence of the search procedure. However, if exploitation is too strong, it can result in the quick loss of diversity in the population and thus may lead to the premature convergence. On the other hand, if new search moves are not guided by local landscape information, it can typically increase the exploration capability and generate new solutions with higher diversity. However, too much diversity and exploration may result in meandered search paths, thus lead to the slow convergence. Therefore, adaptation of search moves so as to balance exploration and exploitation is crucial. Consequently, to maintain the balanced diversity in a population is also important.

Diversity in meta-heuristic algorithms can also appear in many forms. The simplest diversity is to allow the variations of solutions in the population by randomization. For example, solution diversity in genetic algorithms is mainly controlled by the mutation rate and crossover mechanisms, while in simulated annealing, diversity is achieved by random walks. In most SI-based algorithms, new solutions are generated according to a set of deterministic equations, which also include some random variables. Diversity is represented by the variations, often in terms of the population variance. Once the population variance is getting smaller (approaching zero), diversity also decreases, leading to converged solution sets. However, if diversity is reduced too quickly, premature convergence may occur. Therefore, a right amount of randomness and the right form of randomization can be crucial.

In summary, adaptation and diversity in meta-heuristic algorithms can mainly take the following forms:

- balance of exploration and exploitation,
- generation of new solutions,
- right amount of randomness,
- parameter setting, and
- other subtle form.

In the remainder of this chapter, we discuss the role of adaptation and diversity in these cases.

5.1 Exploration and Exploitation

The efficiency of a search process in all population-based nature-inspired algorithms depends on two components: *exploration* and *exploitation* [21]. The first component is connected with the generation of new undiscovered regions of the search space, while the second with directing the search towards the known good solutions. Both components must be balanced during the search because too much exploration can lead to inefficient search, while too much exploitation can lead to the loss of the population diversity that may cause *premature convergence*. Exploitation and exploration are also referred to as intensification and diversification [59,176,10].

Exploitation uses any information obtained from the problem of interest so as to help to generate new solutions that are better than existing solutions. However, this process is typically local, and information (such as gradients) is

also local. Actually, it is for a local search. For example, hill-climbing is a method that uses derivative information to guide the search procedure. In fact, new steps always try to climb up the local gradient. The advantage of exploitation is that it usually leads to very high convergence rates, but its disadvantage is that it can get stuck in a local optimum because the final solution point largely depends on the starting point.

On the other hand, exploration makes it possible to explore the search space more efficiently, and it can generate solutions with enough diversity and far from the current solutions. Therefore, the search is typically on a global scale. The advantage of exploration is that it is less likely to get stuck in a local mode, and the global optimality can be more accessible. However, its disadvantages are slow convergence and waste of a lot of computational efforts because many new solutions can be far from global optimality.

As a result, a fine balance is required so that an algorithm can achieve the best performance. Too much exploitation and too little exploration means the system may converge more quickly, but the probability of finding the true global optimality may be low. On the other hand, too little exploitation and too much exploration can cause the search path meander with very slow convergence. The optimal balance should mean the right amount of exploration and exploitation, which may lead to the optimal performance of an algorithm. Therefore, a proper balance is crucially important.

However, how to achieve such a balance is still an open problem. In fact, no algorithm can claim to have achieved such an optimal balance in the current literature. In essence, the balance itself is a hyper-optimization problem, because it is the optimization of an optimization algorithm. In addition, such a balance may depend on many factors such as the working mechanism of an algorithm, its setting of parameters, tuning and control of these parameters and even the problem to be considered. Furthermore, such a balance may not universally exist [18], and it may vary from problem to problem, thus requiring an adaptive strategy.

These unresolved problems and mystery can motivate more research in this area, and it can be expected relevant literature will increase in the near future.

Attraction and Diffusion. The novel idea of attraction via light intensity as an exploitation mechanism was first used by Yang in the firefly algorithm (FA) in 2007 and 2008. It is simple, flexible and easy to implement. This algorithm bases on the flashing patterns and behavior of tropical fireflies, and can naturally deal with nonlinear multimodal optimization problems.

The movement of firefly i is attracted to another more attractive (brighter) firefly j as determined by

$$\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \beta_0 e^{-\gamma r_{ij}^2}(\mathbf{x}_j^{(t)} - \mathbf{x}_i^{(t)}) + \alpha\, \boldsymbol{\epsilon}_i^{(t)}, \tag{10}$$

where the second term is due to the attraction, and β_0 is the attractiveness at $r = 0$. The third term is randomization with α being the randomization parameter, and $\boldsymbol{\epsilon}_i^{(t)}$ is a vector of random numbers drawn from a Gaussian distribution

at time t. Other studies also use the randomization in terms of $\boldsymbol{\epsilon}_i^{(t)}$ that can easily be extended to other distributions such as Lévy flights. A comprehensive review of the firefly algorithm and its variants has been carried out by Fister et al. [74,79,75].

In FA, the attractiveness (and light intensity) is intrinsically linked with the inverse-square law of light intensity variations and the absorption coefficient. As a result, there is a novel but nonlinear term of $\beta_0 \exp[-\gamma r^2]$ where β_0 is the attractiveness at the distance $r = 0$, and $\gamma > 0$ is the absorption coefficient for light [10].

The main function of such attraction is to enable an algorithm to converge quickly because these multi-agent systems evolve, interact and attract, leading to some self-organized behavior and attractors. As the swarming agents evolve, it is possible that their attractor states will move towards to the true global optimality.

This novel attraction mechanism in FA is the first of its kind in the literature of nature-inspired computation and computational intelligence. This also motivated and inspired others to design similar or other kinds of attraction mechanisms. Other algorithms that were developed later also used inverse-square laws, derived from nature. For example, the charged system search (CSS) used Coulomb's law, while the gravitational search algorithm (GSA) used Newton's law of gravitation.

Whatever the attraction mechanism may be, from the meta-heuristic point of view, the fundamental principles are the same: that is, they allow the swarming agents to interact with one another and provide a forcing term to guide the convergence of the population.

Attraction mainly provides the mechanisms for exploitation, but, with proper randomization, it is also possible to carry out some degree of exploration. However, the exploration is better analyzed in the framework of random walks and diffusive randomization. From the Markov chain point of view, random walks and diffusion are both Markov chains. In fact, Brownian diffusion such as the dispersion of an ink drop in water is a random walk. Lévy flights can be more effective than standard random walks. Therefore, different randomization techniques may lead to different efficiency in terms of diffusive moves. In fact, it is not clear what amount of randomness is needed for a given algorithm.

5.2 Generation of New Solutions

The ways of generating new solutions affect the performance of an algorithm. There are as many ways of solution generations as the number of variants or algorithms. For example, according to Yang [173], three major ways of generating the new solutions in SI-based algorithms are:

– Uniform random generation between a lower bound $\mathbf{L}$ and an upper bound $\mathbf{U}$. Thus, the new solution often takes the form

$$\mathbf{x} = \mathbf{L} + \epsilon(\mathbf{U} - \mathbf{L}), \tag{11}$$

where $\epsilon \in [0, 1]$.

– Local random walks around a current solution (often the best solution), which gives

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + w, \tag{12}$$

where w is drawn from a Gaussian normal distribution.

– Global Lévy flights provide an efficient way of generating long-jump solutions

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + L(\lambda), \tag{13}$$

where $L(\lambda)$ obeys a Lévy distribution with the exponent of λ.

However, it is very rare for an algorithm to use only one of the above methods. In fact, most algorithms use a combination of the above methods together with other ways of solution generation.

5.3 Right Amount of Diversity via Randomization

As we mentioned earlier, all meta-heuristic algorithms have to use stochastic components (i.e., randomization) to a certain degree. Randomness increases the diversity of the solutions and thus enables an algorithm to have the ability to jump out of any local optimum. However, too much randomness may slow down the convergence of the algorithm and thus can waste a lot of computational efforts. Therefore, there is some tradeoff between deterministic and stochastic components, though it is difficult to gauge what is the right amount of randomness in an algorithm? In essence, this question is related to the optimal balance of exploration and exploitation, which still remains an open problem.

As random walks are widely used for randomization and local search in meta-heuristic algorithms [10,9], a proper step size is very important. As different algorithms use different forms of randomization techniques, it is not possible to provide a general analysis for assessing randomness.

One of the simplest randomization techniques is probably the so-called random walk, which can be represented as the following generic equation

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + s\boldsymbol{\epsilon}^{(t)}, \tag{14}$$

where $\boldsymbol{\epsilon}^{(t)}$ is drawn from a standard normal distribution with a zero mean and unity standard deviation. Here, the step size s determines how far a random walker (e.g., an agent or a particle in meta-heuristics) can go for a fixed number of iterations. Obviously, if s is too large, then the new solution $\mathbf{x}^{(t+1)}$ generated will be too far away from the old solution (or more often the current best). Then, such a move is unlikely to be accepted. If s is too small, the change is too small to be significant, and consequently such search is not efficient. So a proper step size is important to maintain the search as efficient as possible. However, what size is proper may depend on the type of the problem and can also be changed during the iteration. Therefore, step sizes and thus the amount of randomness may have to be adaptive.

5.4 Parameter Settings in Computational Intelligence

Biological species live in a dynamic environment. When the environment changes these changes are also followed by living beings changing their behavior as determined by corresponding genetic material written in chromosomes. Those beings who do not follow these changes are eliminated by the natural selection. The extinction of mammoths is a well-known example of animals that were not capable of adapting to new environmental conditions that occurred after the recent Ice Age.

On the other hand, the changing environment of the Galápagos archipelago essentially influenced the adaptive radiation of Darwin's finches. At that time, some islands had disappeared, while some new ones had emerged because of volcanic activity within that region. The tropical climate from before the onset of the recent Ice Age had changed by global cooling that crucially influenced the vegetation. Consequently, the ancestral finches acquired longer and narrower beaks better suited to exploring for nectar and insects [3] thus changing their habitat regarding trees by living on the ground. In line with this, the ground finches had also changed their feeding habits, i.e., in place of nectar and insect they fed on seeds. Those ground finches with the shorter beaks were more suitable for this living space and therefore had more chances of surviving and reproducing their genetic material for the next generations. Additionally, mutations were ensured for the modification of this material, where only successful mutations ensured individuals survived.

In summary, it can be concluded that finches adapted to a changing environment with their body size and shape of their beaks. Both characteristics are written in chromosomes that were changed via crossover and mutation. As matter of fact, the adaptation process can be viewed from almost three aspects to: when to adapt (environment), what to adapt (chromosomes), and how to adapt (crossover and mutation).

How can we use this adaptation metaphor from biology in computational intelligence (CI)? As stated previously, a problem in EAs relates to the environment in nature. However, this formulation can also be widened to other population-based CI algorithms. If the problem is solved by an algorithm, then its behavior is determined by the algorithm parameters. In other words, the algorithm parameters (also strategic parameters) control the behavior of the algorithm.

For instance, EAs have several parameters like the probability of crossover, probability of mutation, etc. [36]. The former regulates the probability that the crossover operator will be applied to two or more parents, while the latter the probability that the mutation will change a generated offspring. The parameters CR and F are used in DE for the same purposes. The other SI-based and ANN algorithms use specific algorithm parameters depending on the biological, physical, chemical, and all other rules that inspire developers of the new algorithms [30].

An instance of parameter values set during the run is also-called a *parameter setting*. Obviously, the different values of parameters, i.e., parameter setting can lead to different results and indirectly to different behavior by an algorithm.

Therefore, it can be concluded that CI algorithms adapt their *parameters* (what?) to *a problem* to be solved (when?) by *changing algorithm parameters* (how?). Links between a natural adaptation and adaptation in CI is made in Table 2, where the adaptation domains are analyzed according to three different aspects, i.e., when to adapt, what to adapt and how to adapt.

Table 2. Adaptation in natural and artificial systems

Adaptation	When?	What?	How?
Natural	Environment	Structures	Operators
ANN	Problem	Perceptrons	Learning
EAs and SI	Problem	Parameter	Changing parameter settings

The adaptation in ANNs is embedded into the algorithm's structures, where perceptrons learn how to minimize the error rate. On the other hand, the population-based CI search algorithms improve the fitness by changing the parameter settings. According to Eiben and Smith [36], the algorithm parameters can be changed:

- deterministically,
- adaptively,
- self-adaptively.

Deterministic parameter control takes place when the strategy parameters are changed by some deterministic rule. That means, this deterministic rule is predefined and therefore any feedback from a search process is not necessary. For instance, parameters can be changed in a time-varying schedule, i.e., when a predefined number of generations have elapsed [36].

Adaptive parameter control means that the strategy parameters are changed according to some form of feedback from the search process. An example of this parameter control is the well-known 1/5 success rule of Rechenberg [51], where the mutation strength (probability of mutation) is increased when the ratio of successful mutation is greater than 1/5 and decreased when the ratio of successful mutation is less than 1/5. In the first case, the search process focuses on exploring the search space, while in the second case on searching around the current solution, i.e., exploiting the search space.

Control parameters are encoded into chromosomes and undergo actions by the variation operators (e.g., crossover and mutation) using self-adaptive parameter control. The better values of parameter variables and control parameters have more chances to survive and reproduce their genetic material into the next generations. This phenomenon makes EAs more flexible and closer to natural evolution [53]. This feature was firstly introduced in ES by Schweffel [52].

Parameter control addresses only one side of the parameter setting, where the strategic parameters are changed during the run. In contrast, when the parameters are fixed during the run, an optimal parameter setting needs to be found by

an algorithm's developer. Typically, these optimal parameters are searched during a *tuning*. In general, the taxonomy of parameter setting according to Eiben and Smith [36] is as illustrated in Fig. 8.

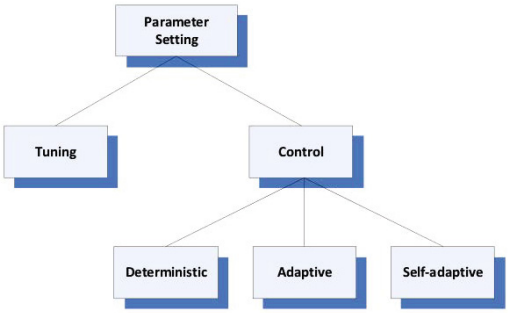


Fig. 8. Parameter setting in CI algorithms

Obviously, the different values of strategic parameters may lead to different results, i.e., the results obtained by one parameter setting can be better than by another and vice versa. In order to find the best parameter setting, the tuning of parameters is performed that demands extensive experimental work. This work can be increased enormously when the algorithm has more parameters to be tuned, and where also an analysis as to how the combination of the individual parameters must be taken into consideration [17].

6 Hybridization in Computational Intelligence

This section deals with a hybridization in CI. Here, we are focused on the nature-inspired CI algorithms. According to their characteristics, two types of the hybridization in CI can be considered, as follows: hybridization in ANNs and hybridization in population-based CI search algorithms. Actually, it is hard to treated both types of hybridizations separately, because the hybridization becomes a powerful bond that connects the individual algorithms under the same umbrella. In line with this, boundaries between individual algorithms composing such the hybrid algorithm are deleted, while the hybrid algorithm operates as a homogenous unit by solving the hardest real-world problems.

In the remainder of the chapter, hybridizations of ANNs and population-based CI search algorithms are presented in details.

6.1 Hybridization in Neural Networks

The hybridization of ANNs with EAs and SI-based algorithms is aimed at solving two optimization problems arising during the application of ANNs. The first

problem arises because gradient-based methods for ANN training are susceptible to getting stuck in local optimums on complex error surfaces. For such cases, global search methods like EAs and SI-based algorithms can provide a robust and efficient approach for weight optimization. The second problem arises because the optimal network structure for a specific task is rarely known in advance and is usually determined by an expert through a tedious experimentation process. When using EA or SI-based algorithm, the network topology can be dynamically adapted to the problem at hand by the insertion and removal of neurons or the connections between them.

The field of *neuro-evolution* provides an unified framework for adaptive evolution and the training of neural networks. In neuro-evolution the ANN structure and weights are adaptively developed using one of the nature-inspired optimization methods with a problem specific fitness function. We can distinguish three groups of neuro-evolutionary methods depending on whether the network parameters (i.e., weights), topology or both, are evolved. Further, because the concept of application to the training and evolution of ANN is very similar using either EAs or SI-based methods, we regard them all under the term of neuro-evolution in this text (Fig. 9).

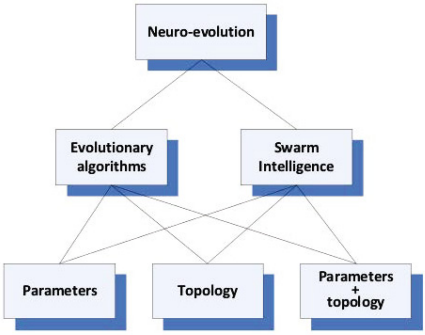


Fig. 9. Hybridization in ANNs

6.2 Hybridization in Population-Based CI Search Algorithms

EAs and SI-based algorithms belong to a class of population-based CI search algorithms. This means, these algorithms maintain a population of solutions in place of a single point solution during the run. While the single point search algorithms deal with single points within a fitness landscape, population-based algorithms investigate the sub-regions of points within the same landscape. Beside this inherent parallelism, the population-based search algorithms are more likely to provide a better balance between the simultaneous exploration of these sub-regions and exploitation of the knowledge accumulated in the representation of the solutions.

As a result, the population-based search algorithms like EAs and SI-based algorithms, rely on balancing exploration and exploitation within the search process [36]. The former is connected with discovering new solutions, while the latter with directing the search process in the vicinity of good solutions. Both components of the search process are controlled indirectly by the algorithms parameters. Therefore, the suitable parameter settings can have a great impact on the performance of the population-based search algorithms. Actually, these algorithms operate correctly, when a sufficient population diversity is present. The population diversity can be measured as: the number of different fitness values, the number of different phenotypes, entropy, and others [21]. The higher the population diversity, the better the exploration of the search space. Losing population diversity leads to premature convergence. In SI, stagnation can also occur where the current best solution can no longer be improved [23].

In general, the population-based search algorithms can be considered as general problem solvers that can be successfully applied to the many NP-hard problems occurring in practice. Unfortunately, the metaphor *general problem solver* does not mean that they obtain the best solution for each of our problems. In this sense, they act similarly to a Swiss Army knife [54] that can be used to address a variety of tasks. Definitely, the majority of tasks can be performed better using the specialized tools but, in absence of these tools, the Swiss Army knife may be a suitable replacement for them. For instance, when slicing a piece of bread, the kitchen knife is more suitable but when traveling the Swiss Army knife is fine.

Although population-based CI algorithms provide adequate solutions for most real-world problems and therefore can even be applied to domains where the problem-specific knowledge is absent, they perform worse when solving the problems from domains where a lot of problem-specific knowledge has to be explored. This is consistent with the so-called No-Free Lunch theorem [18] arguing that any two algorithms are equivalent when their average performances are compared across all classes of problems. This theorem that in fact destroys our dreams about developing a general problem solver can fortunately be circumvented for a specific problem by hybridizing, i.e., incorporating problem-specific knowledge into the algorithm. However, no exact solutions of the problems are needed, in practice, and therefore the primary task is to find the efficient tool for solving a specific class of problems effectively.

On the one hand, integration of population-based search algorithms with one or more refinement methods in order to conduct problem-specific knowledge within the stochastic search process, represents a synergistic combination that often enhances the performance of the population-based search algorithms [24]. On the other hand, this synergistic combination of population-based search and refinement methods is capable of better balancing between exploration and exploitation within the stochastic search process. Obviously, the population-based search is more explorative, while the refinement methods act more exploitatively. Mostly, the refinement methods address the following elements of the population-based search algorithms [55]:

– initial population,
– genotype-phenotype mapping,
– evaluation function, and
– variation and selection operators.

This chapter has focused on population-based CI search algorithms composed within the evolutionary framework. In line with this, the typical refinement methods applied within this class of algorithms are as follows:

– automatic parameter tuning,
– hybridization of components,
– construction heuristics,
– local search heuristics (also memetic algorithms [19,20]).

In the remainder of the chapter, these refinement methods are illustrated in detail. This section concludes with a case study, that presents how hybridization can be performed in typical EAs.

Automatic Parameter Tuning. As an algorithm is a set of interacting Markov chains, we can in general write an algorithm as

$$\begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{pmatrix}^{(t+1)} = A[\mathbf{x}_1, ..., \mathbf{x}_n, p_1, ..., p_k, \epsilon_1, ..., \epsilon_m] \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{pmatrix}^{(t)}, \tag{15}$$

which generates a set of new solutions $(\mathbf{x}_1, ..., \mathbf{x}_n)^{(t+1)}$ from the current population of n solutions. This behavior of an algorithm is largely determined by the eigenvalues of the matrix A that are in turn controlled by the parameters $\mathbf{p} = (p_1, \ldots, p_k)$ and the randomness vector $\boldsymbol{\epsilon} = (\epsilon_1, ..., \epsilon_m)$. From the Markovian theory, we know that the first largest eigenvalue is typically 1, and therefore the convergence rate of an algorithm is mainly controlled by the second largest eigenvalue $0 \leq \lambda_2 < 1$ of A. However, it is extremely difficult to find this eigenvalue in general. Therefore, the tuning of parameters becomes a very challenging task.

The parameter tuning can be defined as an optimization problem that searches for those values of the strategic parameters that optimize the performance of the population-based CI search algorithm [17]. In fact, parameter tuning, or tuning of parameters, is an important topic under active research [17,177]. The aim of parameter tuning is to find the best parameter setting so that an algorithm can perform most efficiently for a wider range of problems. At the moment, parameter tuning is mainly carried out by detailed, extensive parametric studies, and there is no efficient method in general. In essence, parameter tuning itself is an optimization problem which requires higher-level optimization methods to tackle. However, a recent study had shown that a framework for self-tuning algorithms can be established with promising results [177].

In summary, studying how the algorithm depends on its parameters is often of interest to the algorithm's designer. However, both mentioned tasks occur by parameter tuning that can be conducted either manually by a designer or automatically by an algorithm. Because the manually parameter setting is time consuming, automatic parameter tuning is increasingly prevailing. Here, a traditional population-based CI search algorithm can be used for automatic tuning. In this approach, one population-based CI search algorithm controls the performance of another by changing its parameter setting, while the other algorithm solves the original problem and therefore works within the corresponding problem space. The control algorithm operates in the parameter space of the controlled algorithm, i.e., at the higher level. Therefore, this approach is also named as meta-heuristic and was introduced by Grefenstette in 1986 [34]. Recently, the word meta-heuristic (meaning 'higher-level' [9]) has become used for any combination of population-based CI search algorithms and appropriate refinement methods.

Hybridization of Components. The EA domain has been matured over more than 50 years of development. Small numbers of problems in science, as well as in practice, remain intact by the evolutionary approach. In line with this, many prominent experts have emerged within this domain together with several original solutions developed by solving this huge diapason of problems. These original solutions were mostly tackled for developing new evolutionary operators, population models, elitism, etc.

Typically, SI-based algorithms borrow the DE operators of mutation and crossover that replace the original move operator in order to increase the efficiency of the SI-based search process. Obviously, the DE variation operators are effective because of their exploration and exploitation power. For instance, Fister et al. in [31] hybridized the BA algorithm with 'DE/rand/1/bin' strategy of applying the mutation and crossover, and reported significant improvements compared with the original BA algorithm, as well as the other well-known algorithms, like ABC, DE and FA.

Construction Heuristics. Usually, population-based CI search algorithms are used for solving those problems where a lot of knowledge has to be accumulated within different heuristic algorithms. Unfortunately, those algorithms operate well on a limited number of problems. On the other hand, population-based CI search algorithms are in general more matured and therefore prepared for solving the various classes of problems, although they suffer from a lack of problem-specific knowledge. In order to combine the advantages of both, population-based CI search algorithms are used for discovering new solutions within the search space, and exploiting these for building new, possibly better solutions.

Construction heuristics build solutions incrementally, i.e., elements are added to the solution step by step until the final solution is obtained (Algorithm 4).

Algorithm 4. Pseudo-code of construction heuristic

```
1: y = ∅
2: while solution y ∈ S not found do
3:     add element y_i ∈ I to solution y heuristic
4:     move the the next element
5: end while
```

Greedy heuristics are the simplest type of construction heuristics that add new elements to a solution according to the value of current heuristic function that can maximize (or minimize) the current non-final set of elements during each construction step. When the stochastic construction heuristics [60] are used, the results of construction may depend on some coincidence. As a result, combining the population-based CI search algorithms which are stochastic in their nature with stochastic construction heuristics form synergy suitable for solving the hardest real-world problems.

Memetic Algorithms. The hybridization of population-based CI search algorithms with local search methods is also named as memetic algorithms (MA). The term MA originated from Moscato in 1989 [56] and means: similar as genes form the "instructions for building proteins" in genetic, memes are "instructions for carrying out behavior, stored in brains" [24]. The term meme was introduced by Dawkins in its famous book *The selfish gene* [58]. In computer science and engineering, a meme represents the smallest piece of knowledge that can be replicated, modified and combined with other memes in order to generate a new meme [22].

Interestingly, there is a difference between the evolution of memes and evolution of genes. While the former does not alter the memetic information at this stage, the latter modified the genetic information during the variation process. However, both changes have their own metaphor in biology. The first can be attributed to the Baldwian model of evolution arguing that behavior characteristics can also be learned during the life-time of individual and therefore not written in genes, while the second is inspired by the Lamarkian model of evolution stating that each behavior characteristics are written in genes.

A local search [59] is an iterative process of investigating the set of points in the neighborhood of the current solution and exchanging it, when a better solution is found [60]. The neighborhood of the current solution $\mathbf{y}$ is defined as a set of solutions achieved by using the elementary operator $\mathcal{N} : S \to 2^S$. All points in neighborhood $\mathcal{N}$ are reached from the current solution $\mathbf{y}$ in k strokes. Therefore, this set of points is also named k-opt neighborhood of point $\mathbf{y}$.

Algorithm 5. Pseudo-code of local search

```
generate initial solution y ∈ S
repeat
    find the next neighbor y' ∈ N(y)
    if f(y') < f(y) then
        f(y) = f(y')
    end if
until neighbor set is empty
```

It should be noticed that MAs represent the simplest class of so-called *meme-inspired computation* (MC) that are also known as *simple hybrids* by Chen et al. in [24]. Recently, MAs have merged with the field of hybridization with adaptation. In line with this, several studies have been emerged that extended the concept of adaptation of parameters also to adaptation of operators [25] that represent the next step in evolution of MC, i.e., *adaptive hybrids*. In contrast to simple hybrids in which domain knowledge is only captured and incorporated once by a human expert during the design of MAs, adaptive hybrids incorporate the adaptive strategies and adaptive parameters in order to better suit to solve the problem as the search process progress [24,26]. To date, the further step of evolution of MC represents the *memetic automation* already described in Section 1 [28]. In the context of MC, all mentioned refinement methods represent the attempt to use memes as the carriers of the various kind of knowledge [27].

Case Study: Hybridization of EAs. Fig. 10 illustrates some possibilities how and where to hybridize EAs. In general, the other population-based CI search algorithms, e.g., SI can also be hybridized in the similar way.

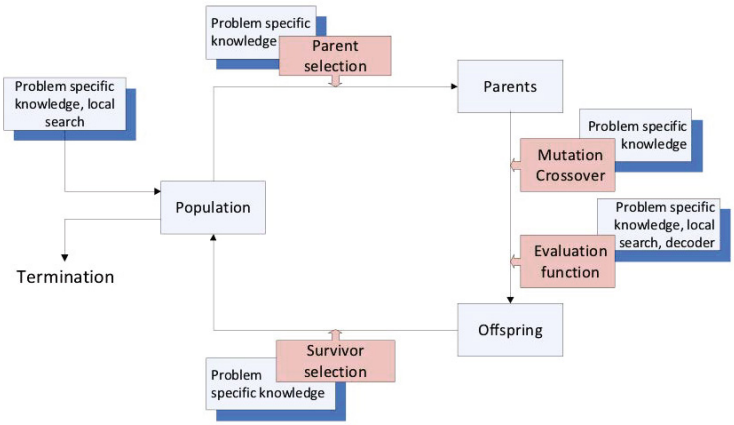


Fig. 10. How to hybridize EAs

At first, the initial population can be generated by incorporating solutions of existing algorithms or by using heuristics, local search, etc. In addition, the local search can be applied to the population of offsprings. Evolutionary operators (e.g., mutation, crossover, parent and survivor selection) can incorporate problem-specific knowledge or apply the operators taken from other algorithms. Finally, a fitness function evaluation offers more possibilities for a hybridization. As a matter of fact, it can be used as a decoder that decodes the indirect represented genotype into a feasible solution. By this mapping, however, various kinds of the problem-specific knowledge or even the traditional heuristics can be incorporated within the algorithm.

7 Applications in Computational Intelligence

Applications of various stochastic population-based CI search algorithms are very diverse, and therefore it is hard to review all the recent developments. In this chapter, we outline some interesting studies briefly.

7.1 Adaptive EAs

EAs were usually connected with parameter adaptation and self-adaptation. Different forms of adaptation and self-adaptation were also applied to the original DE in order to improve its performance. For instance, Qin and Suganthan [105] developed a self-adaptive DE (SaDE). In this version, learning strategy and DE control parameters F and CR are not demanded to be known in advance. That means, learning strategy and parameters are self-adapted during the run according to the learning experience. Brest et al. [64] proposed a DE variant called jDE. Here, control parameters are self-adaptively changed during the evolutionary process. Another variant of self-adaptive DE with the neighborhood search was proposed by Yang [112]. GAs also encompass enormous work in adaptation and self-adaptation domain. In line with this, a very interesting work was proposed by Hinterding et al. [89] that self-adapts mutation strengths and population size. Deb and Beyer [68] developed a self-adaptive GA with simulated binary crossover (SBX). A more complete reviews of the other works in this domain can also be found in [32,33,65,69,110].

7.2 Adaptive SI-Based Algorithms

Adaptations in SI were used less frequently than hybridizations. Usually, adaptation is connected with the adaptation and self-adaptation of control parameters, mutation strategies, learning and etc. Some adaptation forms of ABC was proposed in order to improve search ability of the algorithm, to avoid local optima, and to speed up convergence. For instance, Liao et al. [97] developed an ABC algorithm and applied it to long-term economic dispatch in cascaded hydropower systems. Furthermore, Pan et al. [103] added a self-adaptive strategy for generating neighboring food sources based on insert and swap operators, which

allow the algorithm to work on discrete spaces. Alam and Islam proposed an interesting ABC variant called artificial bee colony with self-adaptive mutation (ABC-SAM) which tries to dynamically adapt the mutation step size with which bees explore the problem search space. In line with this, small step sizes serve to an exploitation component, while large mutation steps more to exploration component of the ABC search process.

On the other hand, some interesting adaptation has also been applied to the bat algorithm (BA). For example, Fister et al. [73] proposed a self-adaptive bat algorithm (SABA), based on the self-adapting mechanism borrowed from the jDE algorithm. In addition, adaptation or self-adaptation in cuckoo search (CS) has yet to be developed.

However, there are some adaptive and self-adaptive variants of the FA. For instance, Fister et al. extended the original FA with the self-adaptation of control parameters called also MSA-FFA and achieved better balancing between exploration and exploitation of the search process. They tested their proposed approach on the graph coloring and showed very promising results [63]. This MSA-FFA was modified by Galvez and Iglesias and adopted for continuous optimization problems [81]. Yu et al. [113] proposed a self-adaptive step FA to avoid falling into the local optimum and reduce the impact of the maximum of generations. Author's core idea was to set the step of each firefly varying with the iteration according to current situation and also historical information of fireflies. Roy et al. [106] developed a FA variant using self-adaptation of the algorithm control parameter values by learning from the fireflies' previous experiences, which led to a more efficient algorithm.

Adaptations in improving PSO are widely spread in many papers describing many applications. Since there are many efficient PSO variants, and readers can refer to the following papers [99,109,115,90,114,95].

7.3 Hybrid ANN+EAs

There is a vast body of literature on the subject of combining EA and ANN, which is nicely assembled in an indexed bibliography [117]. Early neuro-evolution approaches focused on ANN training and demonstrated superior efficiency of EA methods over traditional back-propagation training in many domains [169]. The shift towards the evolution of network topology required consideration of efficient encoding schemes to resolve the problem of multi-way genotype to phenotype maps and avoid small genotypic mutations to result in vastly different phenotypes [140,164]. Most of the work in the last two decades concentrated on the simultaneous evolution of both weights and topology, where various paradigms of EAs have been employed for the evolution of neural networks.

For example, Angeline et al. proposed an approach based on evolutionary programming (EP) to build recurrent neural networks [119]. A similar EP-based method for feed-forward ANNs was presented by Yao and Liu [165]. More recently, Oong and Isa described a hybrid evolutionary ANN (HEANN) in which both the weights and topology were evolved using an adaptive EP method [151].

The symbiotic adaptive neuro-evolution (SANE) by Moriarty used cooperative coevolution to evolve neural networks that adapt to input corruption [145,133].

NeuroEvolution of Augmenting Topologies (NEAT) is an approach that evolves the network topology and adjusts the weights using the genetic algorithm [160,159]. Later, Stanley introduced the HyperNEAT which used compositional pattern producing network as a developmental encoding scheme and was aimed at evolving large neural networks [158]. HyperNEAT is able to capture symmetries in the geometric representation of the task and was extended by Risi into Evolvable Substrate HyperNEAT (ES-HyperNEAT) which added adaptive density of hidden neurons [152]. Evolution of adaptive networks using improved developmental encoding that outperformed HyperNEAT was proposed by Suchorzewski [161]. A multi-objective approach to the evolution of ART networks with adaptive parameters for the genetic algorithm was proposed in a PhD thesis by Kaylani [135]. Hierarchical genetic algorithms, which used parametric and control genes to construct the chromosome, were applied for neuroevolution by Elhachmi and Guennoun [126]. On the side of ANN training procedures the focus is in recent years on novel combinations of GA with gradient-based or local optimization methods, which were used to address the problem of stock market time-series prediction [120] and optimize multi-objective processes in material synthesis [128].

Evolution strategy (ES) was regarded as a driving mechanism of ANN evolution by Matteucci [141]. In place of ES, Igel used evolution strategies with adaptive covariance matrix (CMA-ES) as the neuroevolutionary method in [131]. Kassahun and Sommer presented an improved method called Evolutionary Acquisition of Neural Topologies (EANT), which used more efficient encoding and balancing exploration/exploitation of useful ANN structures [134].

Adaptive differential evolution (ADE) is among the most recent methods to train multi-layer ANNs, used by Silva [124], Slowik [157], and Sarangi et al. [153]. Memetic variants of DE were used to solve prediction problems in medicine and biology [127,122]. Cartesian genetic programming was used by several authors to efficiently encode evolvable ANN [137,136,162].

7.4 Hybrid ANN+SI

More recently the ANNs have been coupled with SI-based algorithms. Particle swarm optimization (PSO) was combined with the classical back-propagation (BP) learning method for the training of feed-forward neural networks by Zhang et al. [167]. Very recently, a similar hybridization of PSO with a simplex optimization method was proposed by Liao et al. [139]. A hybrid of PSO and gravitational search algorithm (GSA) outperformed each individual method in ANN training benchmarks [144]. Sermpinis et al. have used the PSO method with adaptive inertia, cognitive, and social factors to improve the performance of a radial basis function (RBF) network in the task of exchange rate forecasting [154]. A similar approach by Zhang and Wu uses adaptive chaotic PSO to train the ANN in a crop classification task [168].

The successful application of PSO in ANN training was followed by the use of other SI-based algorithms. A hybrid of BP and ACO algorithm was used in

ANN for financial forecasting [129]. The domain of stock forecasting attracted researchers who hybridized ANNs with the ABC algorithm [150] and the fish algorithm [156]. A related application of ABC to earthquake time-series prediction is due to Shah et al. [155].

Additionally, for the most recent SI-based algorithms, adaptive hybridizations of ANNs with the FA [142,146], the BA [148], the CS [147], and hunting algorithm/harmony search combination [138] have also been carried out with good results.

ANN training was also approached using the population-based algorithms which are not strictly nature-inspired, such as magnetic optimization algorithm [143], chemical reaction optimization [166], and artificial photosynthesis and phototropism [123].

While the majority of hybrid ANN+SI-based approaches are concerned with ANN training, evolution of both weights and topology using the PSO was presented by Garro et al. [130] and by Ali [118]. A version of PSO called jumping PSO was recently used by Ismail and Jeng to obtain self-evolving ANN [132].

7.5 Hybrid EAs

There are many hybrid variants of EAs. Most studies in this domain are based on hybridization with local search, and recently also on borrowing some principles from SI. In line with this, Grimaccia et al. [84] combined properties of PSO and GA, and tested performance on the optimization of electromagnetic structures. Galinier and Hao in [80] proposed a hybrid EA (HEA) for graph coloring. Their algorithms combined a highly specialized crossover operators with Tabu search [83]. GA-EDA [104] is a good example of a hybrid EA which uses genetic and estimation of distribution algorithms. Niknam [102] developed a new EA algorithm called DPSO-HBMO, which based on the combination of honey bee mating optimization [87] and discrete PSO [171]. Lin [98] proposed a new EA combining DE with the real-valued GA.

7.6 Hybrid SI-Based Algorithms

In order to improve original SI-based algorithms, researchers usually hybridized these with other meta-heuristics, different local searches, fuzzy logic, machine learning methods and other mathematical principles. This chapter has briefly summarized some SI-based hybrids.

ACO has been hybridized in many applications. For instance, Chitty and Hernandez [67] developed a hybrid technique which added the principles of dynamic programming to ACO for solving the problem of dynamic vehicle routing. On the other hand, Wang et al. [107] proposed a hybrid routing algorithm mobile ad hoc network which based on ACO and zone routing framework of bordercasting. Hybrid ACO was also applied to cope with well-known problem a jobshop scheduling in the study [88]. Moreover, Duan and Yu [70] applied hybrid ACO using memetic algorithm for solving the traveling salesman problem.

ABC was also hybridized in many papers to enhance its performance and efficiency. Duan et al. [71] proposed an ABC and quantum EA, where the ABC was adopted to increase the local search capacity and also the randomness of populations. Data clustering was improved with hybrid ABC (HABC) [111], where authors introduced crossover operator of genetic algorithm to ABC and enhance information exchange between bees. Large-scale global optimization was tackled by memetic ABC (MABC) algorithm [72], where the original ABC was hybridized with two local search heuristics: the Nelder-Mead algorithm (NMA) and the random walk with direction exploitation (RWDE) in order to obtain the better balance between exploration and exploitation. Moreover, a hybrid simplex ABC algorithm (HSABC) which combines NMA with ABC was proposed and applied for solving the inverse analysis problems [92]. An interesting hybrid variant of ABC was also applied to solve graph coloring problems [77].

BA has also been developed many hybrid variants, which try to enhance the efficiency, performance, quality of solutions, and faster convergence. A hybrid BA with path relinking was proposed by Zhou et al. [116], where authors integrated the greedy randomized adaptive search procedure (GRASP) and path relinking into the BA, and applied to capacitated vehicle routing problem. Fister et al. [76] created a hybrid BA (HBA) in order to combine the original BA with DE strategies as a local search instead of classic random walk. An extension of the SABA was done by the same authors in [31] where they hybridized the SABA (HSABA) also with ensemble DE strategies that were used as a local search for improving current best solution directing the swarm of a solution towards the better regions within a search space. Wand and Guo developed a novel hybrid BA with harmony search and applied to global numerical optimization [85].

Chandrasekaran and Simon [66] proposed a hybrid CS (HCS) algorithm that was integrated with a fuzzy system in order to cope with multi-objective unit commitment problems. Layeb [94] developed a novel quantum inspired CS that connects the original CS with quantum computing principles. The main advantage of this hybridization was a good balance between exploration and exploitation during the search process. Li and Yin [96] created a new hybrid variant of CS called CS-based memetic algorithm and applied it for solving permutation flow shop scheduling problems. Since the creation of CS, a diverse range of hybrid variants this algorithm have emerged. Therefore, readers are invited to read the review of these algorithms in the paper [91].

FA is another example of very successful SI-based algorithm that experienced many promising hybridizations since its birth in 2008. Although a comprehensive description of this algorithm was performed in papers [74,75], let us present some efficient and recent hybrid variants of the FA only. Kavousi-Fard et al. [93] combined a support vector machine (SVM) and modified FA in order to get a hybrid prediction algorithm and applied it to the short term electrical load forecast. Guo et al. in [86] combined FA with harmony search. The result of this hybridization was an effective algorithm for solving the global numerical optimization problems. On the other hand, Fister et al [78] developed a memetic FA (MFFA) and applied it to the graph coloring problems. Interesting approach to

the distributed graph coloring problem based on the calling behavior of Japanese tree frogs were accomplished by Hernández and Blum in [170].

PSO underwent many hybridization suitable for continuous and combinatorial optimization. For instance, Lovbjerg et al. [100] created a hybrid PSO and borrowed some concepts from EAs. A very interesting method was proposed by Marinakis and Marinaki [101] where authors developed new approach based on PSO, greedy randomized adaptive search procedure and expanding neighborhood search. This algorithm was then tested on the probabilistic traveling salesman problem. Zhang et al. proposed a DEPSO algorithm [172], which combined PSO with DE operators, while Wang and Li [108] combined PSO with simulated annealing (SA).

Obviously, there are other developments and applications, but the purpose of this chapter is not to review all of they. Therefore, interested readers can refer to more specialized literature.

8 Conclusion

Adaptation becomes the metaphor for reactions of the natural or artificial system to the conditions of the changing environment. There are a lot of renewed interests in this area. Therefore, this chapter starts from a definition of adaptive systems and identifies the human domains that already deal with this phenomenon. Adaptation has also been encountered in the domain of problem-solving. In order to solve these problems, developers usually try to develop new algorithms imitating the main characteristics of natural processes. Interestingly, the nature does not impose questions only, but also provides the answers how to solve these. However, these answers provides diverse sources of inspiration for scientists in order to solve their problems.

Researchers have always been trying to find the general problem solver suitable to solve all classes of the real-world problems. However, this is usually not possible as constrained by the NFL theorem. Hybridization of nature-inspired algorithms may partly overcome the limitations of the NFL theorem, when solving a specific problem by incorporating the problem-specific knowledge in the algorithm structures. In line with this, some popular hybridization methods have been presented in the chapter, with emphasize on the memetic algorithms. This initial idea of hybridizing the population-based CI nature-inspired algorithms with the local search has led to the emergence of the new area in CI, i.e., memetic computation that represents the class of new general problem solvers suitable for solving the hardest real-world problems.

Here, we have identified three main sources of inspiration that are the most commonly used nowadays for the development of the new nature-inspired algorithms, i.e., human brains, a Darwinian natural selection, and behavior or some social living insects and animals. In line with this, three classes of nature-inspired algorithms have been emerged, in general: ANNs, EAs and SI-based. All the mentioned classes of algorithms placed under the umbrella of CI are described in detail throughout this chapter. The descriptions of these algorithms

are emphasized in terms of adaptation and hybridization that can be applied in order to increase their performance. At the end, the papers tackling the recent advances in this CI domains are reviewed shortly.

In summary, we hope that this chapter (and the chapters in the book) contains a sufficient information to inspire researchers to begin searching for solutions in the beautiful dynamic world represented by the adaptation and hybridization in CI.

References

1. Beni, G., Wang, J.: Swarm Intelligence in Cellular Robotic Systems. In: Proceedings of NATO Advanced Workshop on Robots and Biological Systems, Tuscany, Italy, pp. 26–30 (1989)
2. Turing, A.M.: Computing machinery and intelligence. Mind, 433–460 (1950)
3. Grant, P.R., Grant, B.R.: Adaptive Radiation of Darwin's Finches. American Scientist 90(2), 130–150 (2002)
4. Wright, S.A.: The roles of mutation, inbreeding, crossbreeding and selection in evolution. In: Proceedings of the VI International Congress of Genetrics, vol. (1), pp. 356–366 (1932)
5. Dasgupta, D.: Information Processing in the Immune System. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 161–167. McGraw Hill, New York (1999)
6. Dorigo, M., Di Caro, G.: The Ant Colony Optimization Meta-heuristic. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 11–32. McGraw Hill, London (1999)
7. Karaboga, D., Bastruk, B.: A Powerful and Efficient Algorithm for Numerical Function Optimization: Artificial Bee Colony (ABC) Algorithm. Journal of Global Optimization 39(3), 459–471 (2007)
8. Kennedy, J., Eberhart, R.: The Particle Swarm Optimization; Social Adaptation in Information Processing. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 379–387. McGraw Hill, London (1999)
9. Yang, X.-S.: A New Metaheuristic Bat-Inspired Algorithm. In: González, J.R., Pelta, D.A., Cruz, C., Terrazas, G., Krasnogor, N. (eds.) NICSO 2010. SCI, vol. 284, pp. 65–74. Springer, Heidelberg (2010)
10. Yang, X.-S.: Firefly Algorithm. In: Yang, X.-S. (ed.) Nature-Inspired Metaheuristic Algorithms, pp. 79–90. Luniver Press, London (2008)
11. Yang, X.-S.: Flower Pollination Algorithm for Global Optimization. In: Durand-Lose, J., Jonoska, N. (eds.) UCNC 2012. LNCS, vol. 7445, pp. 240–249. Springer, Heidelberg (2012)
12. Yang, X.-S., Deb, S.: Cuckoo Search via Levy Flights. In: World Congress & Biologically Inspired Computing (NaBIC 2009), pp. 210–214. IEEE Publication (2009)
13. Storn, R., Price, K.: Differential Evolution: A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization 11(4), 341–359 (1997)
14. Moore, G.E.: Cramming more components onto integrated circuits. Electronics 38(8), 114–117 (1965)
15. Ulam, S.: Tribute to John von Neumann. Bulletin of the American Mathematical Society 64(3), 50–56 (1958)

16. McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. The Bulletin of Mathematical Biophysics 5(4), 115–133 (1943)
17. Eiben, A.E., Smith, S.K.: Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm and Evolutionary Computation 1(1), 19–31 (2011)
18. Wolpert, D., Macready, W.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1(1), 67–82 (1997)
19. Moscato, P.: Memetic algorithms: A short introduction. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 219–234. McGraw Hill, London (1999)
20. Wilfried, J.: A general cost-benefit-based adaptation framework for multimeme algorithms. Memetic Computing 2, 201–218 (2010)
21. Črepinšek, M., Liu, S.-H., Mernik, M.: Exploration and exploitation in evolutionary algorithms: A survey. ACM Computing Surveys 45(3), 1–33 (2013)
22. Neri, F., Cotta, C.: Memetic algorithms and memetic computing optimization: A literature review. Swarm and Evolutionary Computation 1(2), 1–14 (2011)
23. Neri, F.: Diversity Management in Memetic Algorithms. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 153–164. Springer, Berlin (2012)
24. Chen, X., Ong, Y.-S., Lim, M.-H., Tan, K.C.: A Multi-Facet Survey on Memetic Computation. Trans. Evol. Comp. 15(5), 591–607 (2011)
25. Ong, Y.-S., Lim, M.-H., Zhu, N., Wong, K.-W.: Classification of adaptive memetic algorithms: a comparative study. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 36(1), 141–152 (2006)
26. Garcia, S., Cano, J.R., Herrera, F.: A memetic algorithm for evolutionary prototype selection: A scaling up approach. Pattern Recogn. 41(8), 2693–2709 (2008)
27. Iacca, G., Neri, F., Mininno, E., Ong, Y.-S., Lim, M.-H.: Ockham's Razor in memetic computing: Three stage optimal memetic exploration. Inf. Sci. 188(4), 17–43 (2012)
28. Ong, Y.-S., Lim, M.H., Chen, X.: Research frontier: memetic computation-past, present & future. Comp. Intell. Mag. 5 2(5), 24–31 (2010)
29. Lynch, A.: Thought as abstract evolution. J. Ideas 2(1), 3–10 (1991)
30. Fister Jr., I., Yang, X.-S., Fister, I., Brest, J., Fister, D.: A brief review of nature-inspired algorithms for optimization. Electrotehnical Review 80(3), 116–122 (2013)
31. Fister, I., Fong, S., Brest, J., Fister Jr., I.: A novel hybrid self-adaptive bat algorithm. The Scientific World Journal, 1–12 (2014)
32. Fister, I., Mernik, M., Filipič, B.: Graph 3-coloring with a hybrid self-adaptive evolutionary algorithm. Comp. Opt. and Appl. 54(3), 741–770 (2013)
33. Fister, I., Mernik, M., Filipič, B.: A hybrid self-adaptive evolutionary algorithm for marker optimization in the clothing industry. Appl. Soft Comput. 10(2), 409–422 (2010)
34. Grefenstette, J.: Optimization of control parameters for genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics 16, 122–128 (1986)
35. Kotler, P., Armstrong, G., Brown, L., Adam, S.: Marketing, 7th edn. Pearson Education Australia/Prentice Hall, Sydney (2006)
36. Eiben, A., Smith, J.: Introduction to Evolutionary Computing. Springer, Berlin (2003)
37. Darwin, C.: On the Origin of Species. Harvard University Press, London (1859)
38. Blum, C., Merkle, D.: Swarm Intelligence. Springer, Berlin (2008)
39. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Prentice Hall, New Jersey (2009)

40. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
41. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. A Bradford Book, Cambridge (1992)
42. Maschler, M., Solan, A., Zamir, S.: Game Theory. Cambridge University Press, Cambridge (2013)
43. Lehn, J.M.: Supramolecular Chemistry: Concepts and Perspectives. VCH Verlagsgeselschaft, Weinheim (1995)
44. Applegate, D.L., Bixby, R.E., Chvătal, V., Cook, W.: The Traveling Salesman Problem. University Press, Princeton (2006)
45. Bondy, J.A., Murty, U.S.R.: Graph Theory. Springer, Berlin (2008)
46. Bäck, T.: Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press, Oxford (1996)
47. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., Boston (1989)
48. Fogel, L., Owens, A., Walsh, M.: Artificial Intelligence through Simulated Evolution. John Willey & Sons, Inc., New York (1966)
49. Koza, J.: Genetic Programming 2 - Automatic Discovery of Reusable Programs. MIT Press, Cambridge (1994)
50. Searle, J.R.: The rediscovery of the mind. MIT Press, Cambridge (1992)
51. Rechenberg, I.: Evolutionsstrategie, Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog, Stuttgart (1973)
52. Schwefel, H.P.: Numerische Optimierung von Computer–Modellen mittels der Evolutionsstrategie. Birkhäuser, Basel (1977)
53. Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms. John Wiley & Sons, Inc., New York (2001)
54. Michalewicz, Z., Fogel, D.: How to solwe it: Modern heuristics. Springer (2004)
55. Michalewicz, Z.: Genetic algorithms + data structures = evolution programs. Springer, Berlin (1992)
56. Moscato, P.: On evolution, search, optimization, genetic algorithm and martial arts: Toward memetic algorithms. Tech. Rep. 826. California Institute of Technology, Pasadena, CA (1989)
57. Yang, X.-S.: Nature-Inspired Metaheuristic Algorithms. Luniver Press (2010)
58. Dawkins, R.: The selfish gene. Oxford University Press, Oxford (1976)
59. Aarts, E., Lenstra, J.K.: Local Search in Combinatorial Optimization. Oxford University Press, Princeton (1997)
60. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Elsevier, Oxford (2005)
61. Blackmore, S.: The Meme Machine. Oxford University Press, New York (1999)
62. Law, A.: Simulation Modeling and Analysis with Expertfit Software. McGraw-Hill, New York (2006)
63. Fister, I., Fister Jr, I., Brest, J., Yang, X.-S.: Memetic firefly algorithm for combinatorial optimization. In: Filipič, B., Šilc, J. (eds.) Bioinspired Optimization Methods and Their Applications: Proceedings of the Fifth International Conference on Bioinspired Optimization Methods and their Applications, BIOMA 2012, pp. 75–86. Jožef Stefan Institute, Ljubljana (2012)
64. Brest, J., Greiner, S., Bošković, B., Mernik, M., Žumer, V.: Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. IEEE Transactions on Evolutionary Computation 10(6), 646–657 (2006)

65. Cai, Z., Peng, Z.: Cooperative coevolutionary adaptive genetic algorithm in path planning of cooperative multi-mobile robot systems. Journal of Intelligent and Robotic Systems 33(1), 61–71 (2002)
66. Chandrasekaran, K., Simon, S.P.: Multi-objective scheduling problem: Hybrid approach using fuzzy assisted cuckoo search algorithm. Swarm and Evolutionary Computation 5, 1–16 (2012)
67. Chitty, D.M., Hernandez, M.L.: A hybrid ant colony optimisation technique for dynamic vehicle routing. In: Deb, K., Tari, Z. (eds.) GECCO 2004. LNCS, vol. 3102, pp. 48–59. Springer, Heidelberg (2004)
68. Deb, K., Beyer, H.-G.: Self-adaptive genetic algorithms with simulated binary crossover. Evolutionary Computation 9(2), 197–221 (2001)
69. Dilettoso, E., Salerno, N.: A self-adaptive niching genetic algorithm for multimodal optimization of electromagnetic devices. IEEE Transactions on Magnetics 42(4), 1203–1206 (2006)
70. Duan, H., Yu, X.: Hybrid ant colony optimization using memetic algorithm for traveling salesman problem. In: IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, ADPRL 2007, pp. 92–95. IEEE (2007)
71. Duan, H.-B., Xu, C.-F., Xing, Z.-H.: A hybrid artificial bee colony optimization and quantum evolutionary algorithm for continuous optimization problems. International Journal of Neural Systems 20(01), 39–50 (2010)
72. Fister, I., Fister Jr., I., Žumer, V., Brest, J.: Memetic artificial bee colony algorithm for large-scale global optimization. In: 2012 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8. IEEE (2012)
73. Fister Jr, I., Fong, S., Brest, J., Fister, I: Towards the self-adaptation in the bat algorithm. In: Proceedings of the 13th IASTED International Conference on Artificial Intelligence and Applications (2014)
74. Fister, I., Fister Jr., I., Yang, X.-S., Brest, J.: A comprehensive review of firefly algorithms. Swarm and Evolutionary Computation (2013)
75. Fister, I., Yang, X.-S., Fister, D., Fister Jr., I.: Firefly algorithm: A brief review of the expanding literature. In: Cuckoo Search and Firefly Algorithm, pp. 347–360. Springer (2014)
76. Fister Jr., I., Fister, D., Yang, X.-S.: A hybrid bat algorithm. arXiv preprint arXiv:1303.6310 (2013)
77. Fister Jr., I., Fister, I., Brest, J.: A hybrid artificial bee colony algorithm for graph 3-coloring. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) EC 2012 and SIDE 2012. LNCS, vol. 7269, pp. 66–74. Springer, Heidelberg (2012)
78. Fister Jr, I., Yang, X.-S., Fister, I., Brest, J.: Memetic firefly algorithm for combinatorial optimization. arXiv preprint arXiv:1204.5165 (2012)
79. Fister, I., Yang, X.-S., Brest, J., Fister Jr., I.: Modified firefly algorithm using quaternion representation. Expert Syst. Appl. 40(18), 7220–7230 (2013)
80. Galinier, P., Hao, J.-K.: Hybrid evolutionary algorithms for graph coloring. Journal of Combinatorial Optimization 3(4), 379–397 (1999)
81. Galvez, A., Iglesias, A.: New memetic self-adaptive firefly algorithm for continuous optimization. International Journal of Bio-Inspired Computation (2014)
82. Geem, Z.W., Kim, J.H., Loganathan, G.: A new heuristic optimization algorithm: harmony search. Simulation 76(2), 60–68 (2001)
83. Glover, F., Laguna, M.: Tabu search. Springer (1999)

84. Grimaccia, F., Mussetta, M., Zich, R.E.: Genetical swarm optimization: Self-adaptive hybrid evolutionary algorithm for electromagnetics. IEEE Transactions on Antennas and Propagation 55(3), 781–785 (2007)
85. Guo, L.: A novel hybrid bat algorithm with harmony search for global numerical optimization. Journal of Applied Mathematics 2013 (2013)
86. Guo, L., Wang, G.-G., Wang, H., Wang, D.: An effective hybrid firefly algorithm with harmony search for global numerical optimization. The Scientific World Journal 2013 (2013)
87. Haddad, O.B., Afshar, A., Marino, M.A.: Honey-bees mating optimization (hbmo) algorithm: a new heuristic approach for water resources optimization. Water Resources Management 20(5), 661–680 (2006)
88. Heinonen, J., Pettersson, F.: Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem. Applied Mathematics and Computation 187(2), 989–998 (2007)
89. Hinterding, R., Michalewicz, Z., Peachey, T.C.: Self-adaptive genetic algorithm for numeric functions. In: Ebeling, W., Rechenberg, I., Voigt, H.-M., Schwefel, H.-P. (eds.) PPSN 1996. LNCS, vol. 1141, pp. 420–429. Springer, Heidelberg (1996)
90. Ismail, A., Engelbrecht, A.P.: The self-adaptive comprehensive learning particle swarm optimizer. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Engelbrecht, A.P., Groß, R., Stützle, T. (eds.) ANTS 2012. LNCS, vol. 7461, pp. 156–167. Springer, Heidelberg (2012)
91. Fister Jr., I., Fister, D., Fister, I.: A comprehensive review of cuckoo search: variants and hybrids. International Journal of Mathematical Modelling and Numerical Optimisation 4(4), 387–409 (2013)
92. Kang, F., Li, J., Xu, Q.: Structural inverse analysis by hybrid simplex artificial bee colony algorithms. Computers & Structures 87(13), 861–870 (2009)
93. Kavousi-Fard, A., Samet, H., Marzbani, F.: A new hybrid modified firefly algorithm and support vector regression model for accurate short term load forecasting. Expert Systems with Applications 41(13), 6047–6056 (2014)
94. Layeb, A.: A novel quantum inspired cuckoo search for knapsack problems. International Journal of Bio-Inspired Computation 3(5), 297–305 (2011)
95. Li, C., Yang, S., Nguyen, T.T.: A self-learning particle swarm optimizer for global optimization problems. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 42(3), 627–646 (2012)
96. Li, X., Yin, M.: A hybrid cuckoo search via lévy flights for the permutation flow shop scheduling problem. International Journal of Production Research 51(16), 4732–4754 (2013)
97. Liao, X., Zhou, J., Zhang, R., Zhang, Y.: An adaptive artificial bee colony algorithm for long-term economic dispatch in cascaded hydropower systems. International Journal of Electrical Power & Energy Systems 43(1), 1340–1345 (2012)
98. Lin, W.-Y.: A ga–de hybrid evolutionary algorithm for path synthesis of four-bar linkage. Mechanism and Machine Theory 45(8), 1096–1107 (2010)
99. Liu, S., Wang, J.: An improved self-adaptive particle swarm optimization approach for short-term scheduling of hydro system. In: International Asia Conference on Informatics in Control, Automation and Robotics, CAR 2009, pp. 334–338. IEEE (2009)
100. Lovbjerg, M., Rasmussen, T.K., Krink, T.: Hybrid particle swarm optimiser with breeding and subpopulations. In: Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2001, pp. 469–476. Citeseer (2001)

101. Marinakis, Y., Marinaki, M.: A hybrid multi-swarm particle swarm optimization algorithm for the probabilistic traveling salesman problem. Computers & Operations Research 37(3), 432–442 (2010)
102. Niknam, T.: An efficient hybrid evolutionary algorithm based on pso and hbmo algorithms for multi-objective distribution feeder reconfiguration. Energy Conversion and Management 50(8), 2074–2082 (2009)
103. Pan, Q.-K., Fatih Tasgetiren, M., Suganthan, P.N., Chua, T.J.: A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. Information Sciences 181(12), 2455–2468 (2011)
104. Peña, J.M., Robles, V., Larrañaga, P., Herves, V., Rosales, F., Pérez, M.S.: GA-EDA: Hybrid evolutionary algorithm using genetic and estimation of distribution algorithms. In: Orchard, B., Yang, C., Ali, M. (eds.) IEA/AIE 2004. LNCS (LNAI), vol. 3029, pp. 361–371. Springer, Heidelberg (2004)
105. Qin, A.K., Suganthan, P.N.: Self-adaptive differential evolution algorithm for numerical optimization. In: The 2005 IEEE Congress on Evolutionary Computation, vol. 2, pp. 1785–1791. IEEE (2005)
106. Roy, A.G., Rakshit, P., Konar, A., Bhattacharya, S., Kim, E., Nagar, A.K.: Adaptive firefly algorithm for nonholonomic motion planning of car-like system. In: 2013 IEEE Congress on Evolutionary Computation (CEC), pp. 2162–2169. IEEE (2013)
107. Wang, J., Osagie, E., Thulasiraman, P., Thulasiram, R.K.: Hopnet: A hybrid ant colony optimization routing algorithm for mobile ad hoc network. Ad Hoc Networks 7(4), 690–705 (2009)
108. Wang, X.-H., Li, J.-J.: Hybrid particle swarm optimization with simulated annealing. In: Proceedings of 2004 International Conference on Machine Learning and Cybernetics, vol. 4, pp. 2402–2405. IEEE (2004)
109. Wang, Y., Li, B., Weise, T., Wang, J., Yuan, B., Tian, Q.: Self-adaptive learning based particle swarm optimization. Information Sciences 181(20), 4515–4538 (2011)
110. Wu, Q., Cao, Y., Wen, J.: Optimal reactive power dispatch using an adaptive genetic algorithm. International Journal of Electrical Power & Energy Systems 20(8), 563–569 (1998)
111. Yan, X., Zhu, Y., Zou, W., Wang, L.: A new approach for data clustering using hybrid artificial bee colony algorithm. Neurocomputing 97, 241–250 (2012)
112. Yang, Z., Tang, K., Yao, X.: Self-adaptive differential evolution with neighborhood search. In: IEEE Congress on Evolutionary Computation, CEC 2008 (IEEE World Congress on Computational Intelligence), pp. 1110–1116. IEEE (2008)
113. Yu, S., Yang, S., Su, S.: Self-adaptive step firefly algorithm. Journal of Applied Mathematics 2013 (2013)
114. Zhan, Z.-H., Zhang, J., Li, Y., Chung, H.-H.: Adaptive particle swarm optimization. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 39(6), 1362–1381 (2009)
115. Zhang, J., Ding, X.: A multi-swarm self-adaptive and cooperative particle swarm optimization. Engineering Applications of Artificial Intelligence 24(6), 958–967 (2011)
116. Zhou, Y., Xie, J., Zheng, H.: A hybrid bat algorithm with path relinking for capacitated vehicle routing problem. Mathematical Problems in Engineering 2013 (2013)
117. Alander, J.T.: An indexed bibliography of genetic algorithms and neural networks
118. Ali, Y.M.B.: Evolving multilayer feedforward neural network using adaptive particle swarm algorithm. Int. J. Hybrid Intell. Syst. 8(4), 185–198 (2011)

119. Angeline, P.J., Saunders, G.M., Pollack, J.B.: An evolutionary algorithm that constructs recurrent neural networks. IEEE Transactions on Neural Networks 5, 54–65 (1994)
120. Asadi, S., Hadavandi, E., Mehmanpazir, F., Nakhostin, M.M.: Hybridization of evolutionary levenberg-marquardt neural networks and data pre-processing for stock market prediction. Knowl.-Based Syst. 35, 245–258 (2012)
121. Caudell, T.P., Dolan, C.P.: Parametric connectivity: Training of constrained networks using genetic algorithms. In: David Schaffer, J. (ed.) Proceedings of the Third International Conference on Genetic Algorithms. Morgan Kaufmann Publishers (1989)
122. Cruz-Ramírez, M., Hervás-Martínez, C., Gutiérrez, P.A., Pérez-Ortiz, M., Briceño, J., de la Mata, M.: Memetic pareto differential evolutionary neural network used to solve an unbalanced liver transplantation problem. Soft. Comput. 17(2), 275–284 (2013)
123. Cui, Z., Yang, C., Sanyal, S.: Training artificial neural networks using appm. IJWMC 5(2), 168–174 (2012)
124. da Silva, A.J., Mineu, N.L., Ludermir, T.B.: Evolving artificial neural networks using adaptive differential evolution. In: Kuri-Morales, A., Simari, G.R. (eds.) IBERAMIA 2010. LNCS, vol. 6433, pp. 396–405. Springer, Heidelberg (2010)
125. Delgado, M., Pegalajar, M.C., Cuéllar, M.P.: Evolutionary training for dynamical recurrent neural networks: an application in finantial time series prediction. Mathware & Soft Computing 13(2), 89–110 (2006)
126. Elhachmi, J., Guennoun, Z.: Evolutionary neural networks algorithm for the dynamic frequency assignment problem. International Journal of Computer Science & Information Technology 3(3), 49–61 (2011)
127. Fernández, J.C., Hervás, C., Martínez-Estudillo, F.J., Gutiérrez, P.A.: Memetic pareto evolutionary artificial neural networks to determine growth/no-growth in predictive microbiology. Appl. Soft Comput. 11(1), 534–550 (2011)
128. Furtuna, R., Curteanu, S., Leon, F.: Multi-objective optimization of a stacked neural network using an evolutionary hyper-heuristic. Appl. Soft Comput. 12(1), 133–144 (2012)
129. Gao, W.: Financial data forecasting by evolutionary neural network based on ant colony algorithm. In: Deng, H., Miao, D., Lei, J., Wang, F.L. (eds.) AICI 2011, Part III. LNCS, vol. 7004, pp. 262–269. Springer, Heidelberg (2011)
130. Garro, B.A., Sossa, H., Vazquez, R.A.: Design of artificial neural networks using a modified particle swarm optimization algorithm. In: Proceedings of the 2009 International Joint Conference on Neural Networks, IJCNN 2009, pp. 2363–2370 (2009)
131. Igel, C.: Neuroevolution for reinforcement learning using evolution strategies. In: Reynolds, R., Abbass, H., Tan, K.C., Mckay, B., Essam, D., Gedeon, T. (eds.) Congress on Evolutionary Computation (CEC 2003), vol. 4, pp. 2588–2595. IEEE (2003)
132. Ismail, A.Z., Jeng, D.S.: SEANN: A Self-evolving Neural Network based on PSO and JPSO algorithms
133. Kala, R., Shukla, A., Tiwari, R.: Modular symbiotic adaptive neuro evolution for high dimensionality classificatory problems. Intelligent Decision Technologies 5(4), 309–319 (2011)
134. Kassahun, Y., Sommer, G.: Efficient reinforcement learning through evolutionary acquisition of neural topologies. In: ESANN, pp. 259–266 (2005)
135. Kaylani, A.: An Adaptive Multiobjective Evolutionary Approach to Optimize Artmap Neural Networks. PhD thesis, Orlando, FL, USA (2008), AAI3335346

136. Khan, M.M., Ahmad, A.M., Khan, G.M., Miller, J.F.: Fast learning neural networks using cartesian genetic programming. Neurocomputing 121, 274–289 (2013)
137. Khan, M.M., Khan, G.M., Miller, J.F.: Evolution of neural networks using cartesian genetic programming. In: IEEE Congress on Evolutionary Computation, pp. 1–8. IEEE (2010)
138. Kulluk, S.: A novel hybrid algorithm combining hunting search with harmony search algorithm for training neural networks. JORS 64(5), 748–761 (2013)
139. Liao, S.-H., Hsieh, J.-G., Chang, J.-Y., Lin, C.-T.: Training neural networks via simplified hybrid algorithm mixing nelder–mead and particle swarm optimization methods. Soft Computing, 1–11 (2014)
140. Mandischer, M.: Representation and evolution of neural networks, pp. 643–649. Springer (1993)
141. Matteucci, M.: ELeaRNT: Evolutionary learning of rich neural network topologies. Technical report, Carnegie Mellon University (2002)
142. Lee, M.-C., Horng, M.-H., Lee, Y.-X., Liou, R.-J.: Firefly Meta-Heuristic Algorithm for Training the Radial Basis Function Network for Data Classification and Disease Diagnosis. InTech (2012)
143. Mirjalili, S., Sadiq, A.S.: Magnetic optimization algorithm for training multi layer perceptron. In: 2011 IEEE 3rd International Conference on Communication Software and Networks (ICCSN), pp. 42–46 (May 2011)
144. Mirjalili, S., Hashim, S.Z.M., Sardroudi, H.M.: Training feedforward neural networks using hybrid particle swarm optimization and gravitational search algorithm. Applied Mathematics and Computation 218(22), 11125–11137 (2012)
145. Moriarty, D., Miikkulainen, R.: Forming neural networks through efficient and adaptive coevolution. Evolutionary Computation 5, 373–399 (1998)
146. Nandy, S., Karmakar, M., Sarkar, P.P., Das, A., Abraham, A., Paul, D.: Agent based adaptive firefly back-propagation neural network training method for dynamic systems. In: 2012 12th International Conference on Hybrid Intelligent Systems (HIS), pp. 449–454 (December 2012)
147. Nawi, N.M., Khan, A., Rehman, M.Z.: Csbprnn: A new hybridization technique using cuckoo search to train back propagation recurrent neural network. In: Herawan, T., Deris, M.M., Abawajy, J. (eds.) Proceedings of the First International Conference on Advanced Data and Information Engineering (DaEng-2013). LNEE, vol. 285, pp. 111–118. Springer, Heidelberg (2014)
148. Nawi, N.M., Rehman, M.Z., Khan, A.: A new bat based back-propagation (BAT-BP) algorithm. In: Swiątek, J., Grzech, A., Swiątek, P., Tomczak, J.M. (eds.) Advances in Systems Science. AISC, vol. 240, pp. 395–404. Springer, Heidelberg (2014)
149. Neruda, R., Slušný, S.: Parameter genetic learning of perceptron networks. In: Proceedings of the 10th WSEAS International Conference on Systems, ICS 2006, pp. 92–97 (2006)
150. Nourani, E., Rahmani, A.-M., Navin, A.H.: Forecasting stock prices using a hybrid artificial bee colony based neural network. In: 2012 International Conference on Innovation Management and Technology Research (ICIMTR), pp. 486–490 (May 2012)
151. Oong, T.H., Isa, N.A.M.: Adaptive evolutionary artificial neural networks for pattern classification. IEEE Transactions on Neural Networks 22(11), 1823–1836 (2011)
152. Risi, S., Stanley, K.O.: Enhancing es-hyperneat to evolve more complex regular neural networks. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011, pp. 1539–1546 (2011)

153. Sarangi, P.P., Sahu, A., Panda, M.: Article: A hybrid differential evolution and back-propagation algorithm for feedforward neural network training. International Journal of Computer Applications 84(14), 1–9 (2013); Published by Foundation of Computer Science, New York, USA
154. Sermpinis, G., Theofilatos, K.A., Karathanasopoulos, A.S., Georgopoulos, E.F., Dunis, C.L.: Forecasting foreign exchange rates with adaptive neural networks using radial-basis functions and particle swarm optimization. European Journal of Operational Research 225(3), 528–540 (2013)
155. Shah, H., Ghazali, R., Nawi, N.M.: Using artificial bee colony algorithm for mlp training on earthquake time series data prediction. CoRR, abs/1112.4628 (2011)
156. Shen, W., Guo, X., Wu, C., Wu, D.: Forecasting stock indices using radial basis function neural networks optimized by artificial fish swarm algorithm. Knowl.-Based Syst. 24(3), 378–385 (2011)
157. Slowik, A.: Application of an adaptive differential evolution algorithm with multiple trial vectors to artificial neural network training. IEEE Transactions on Industrial Electronics 58(8), 3160–3167 (2011)
158. Stanley, K.O., D'Ambrosio, D.B., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. Artif. Life 15(2), 185–212 (2009)
159. Stanley, K.O., Miikkulainen, R.: Efficient reinforcement learning through evolving neural network topologies. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2002, pp. 569–577 (2002)
160. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. 10(2), 99–127 (2002)
161. Suchorzewski, M.: Evolving scalable and modular adaptive networks with developmental symbolic encoding. Evolutionary Intelligence 4(3), 145–163 (2011)
162. Turner, A.J., Miller, J.F.: Cartesian genetic programming encoded artificial neural networks: A comparison using three benchmarks. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, GECCO 2013, pp. 1005–1012 (2013)
163. Vogl, T.P., Mangis, J.K., Rigler, A.K., Zink, W.T., Alkon, D.L.: Accelerating the convergence of the back-propagation method. Biological Cybernetics 59(4-5), 257–263 (1988)
164. Whitley, D., Starkweather, T., Bogart, C.: Genetic algorithms and neural networks: optimizing connections and connectivity. Parallel Computing 14(3), 347–361 (1990)
165. Yao, X., Liu, Y.: A new evolutionary system for evolving artificial neural networks. IEEE Transactions on Neural Networks 8, 694–713 (1996)
166. Yu, J.J.Q., Lam, A.Y.S., Li, V.O.K.: Evolutionary artificial neural network based on chemical reaction optimization. In: IEEE Congress on Evolutionary Computation, pp. 2083–2090. IEEE (2011)
167. Zhang, J.-R., Zhang, J., Lok, T.-M., Lyu, M.R.: A hybrid particle swarm optimization-back-propagation algorithm for feedforward neural network training. Applied Mathematics and Computation 185(2), 1026–1037 (2007)
168. Zhang, Y., Wu, L.: Crop classification by forward neural network with adaptive chaotic particle swarm optimization. Sensors 11(5), 4721–4743 (2011)
169. Montana, D.J., Davis, L.: Training feedforward neural networks using genetic algorithms. In: Proceedings of the 11th International Joint Conference on Artificial intelligence (IJCAI 1989), vol. 1, pp. 762–767. Morgan Kaufmann Publishers Inc., San Francisco (1989)
170. Hernández, H., Blum, C.: Distributed graph coloring: an approach based on the calling behavior of Japanese tree frogs. Swarm Intelligence, 117–150 (2012)

171. Chen, W.-N., Zhang, J., Chung, H.S.H., Zhong, W.-L., Wu, W.-G., Shi, Y.-H.: A novel set-based particle swarm optimization method for discrete optimization problems. Trans. Evol. Comp. 14, 278–300 (2010)
172. Zhang, W.-J., Xie, X.-F.: DEPSO: Hybrid Particle Swarm with Differential Evolution Operator. IEEE International Conference on Systems, Man and Cybernetics 4, 3816–3821 (2003)
173. Yang, X.S.: Nature-Inspired Optimization Algorithms. Elsevier, London (2014)
174. Ashby, W.R.: Princinples of the self-organizing sysem. In: Von Foerster, H., Zopf Jr., G.W. (eds.) Pricinples of Self-Organization: Transactions of the University of Illinois Symposium, pp. 255–278. Pergamon Press, London (1962)
175. Booker, L., Forrest, S., Mitchell, M., Riolo, R.: Perspectives on Adaptation in Natural and Artificial Systems. Oxford University Press, Oxford (2005)
176. Blum, C., Roli, A.: Metaheuristics in combinatorial optimisation: Overview and conceptural comparision. ACM Comput. Surv. 35, 268–308 (2003)
177. Yang, X.S., Deb, S., Loomes, M., Karamanoglu, M.: A framework for self-tuning optimization algorithm. Neural Computing and Applications 23(7-8), 2051–2057 (2013)