Information Sciences 237 (2013) 82–117

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A survey on optimization metaheuristics

Ilhem Boussaïd [a], Julien Lepagnot [b], Patrick Siarry [b,*]

[a] Université des sciences et de la technologie Houari Boumediene, Electrical Engineering and Computer Science Department, El-Alia BP 32 Bab-Ezzouar, 16111 Algiers, Algeria
[b] Université Paris Est Créteil, LiSSi, 61 avenue du Général de Gaulle 94010 Créteil, France

ARTICLE INFO

Article history:
Received 10 February 2012
Received in revised form 17 December 2012
Accepted 26 February 2013
Available online 7 March 2013

Keywords:
Population based metaheuristic
Single solution based metaheuristic
Intensification
Diversification

ABSTRACT

Metaheuristics are widely recognized as efficient approaches for many hard optimization problems. This paper provides a survey of some of the main metaheuristics. It outlines the components and concepts that are used in various metaheuristics in order to analyze their similarities and differences. The classification adopted in this paper differentiates between single solution based metaheuristics and population based metaheuristics. The literature survey is accompanied by the presentation of references for further details, including applications. Recent trends are also briefly discussed.

© 2013 Elsevier Inc. All rights reserved.

1. Introduction

We roughly define hard optimization problems as problems that cannot be solved to optimality, or to any guaranteed bound, by any exact (deterministic) method within a "reasonable" time limit. These problems can be divided into several categories depending on whether they are continuous or discrete, constrained or unconstrained, mono or multi-objective, static or dynamic. In order to find satisfactory solutions for these problems, metaheuristics can be used. A metaheuristic is an algorithm designed to solve approximately a wide range of hard optimization problems without having to deeply adapt to each problem. Indeed, the greek prefix "meta", present in the name, is used to indicate that these algorithms are "higher level" heuristics, in contrast with problem-specific heuristics. Metaheuristics are generally applied to problems for which there is no satisfactory problem-specific algorithm to solve them. They are widely used to solve complex problems in industry and services, in areas ranging from finance to production management and engineering.

Almost all metaheuristics share the following characteristics: they are nature-inspired (based on some principles from physics, biology or ethology); they make use of stochastic components (involving random variables); they do not use the gradient or Hessian matrix of the objective function; they have several parameters that need to be fitted to the problem at hand.

In the last thirty years, a great interest has been devoted to metaheuristics. We can try to point out some of the steps that have marked the history of metaheuristics. One pioneer contribution is the proposition of the simulated annealing method by Kirkpatrick et al. in 1982 [150]. In 1986, the tabu search was proposed by Glover [104], and the artificial immune system was proposed by Farmer et al. [83]. In 1988, Koza registered his first patent on genetic programming, later published in 1992 [154]. In 1989, Goldberg published a well known book on genetic algorithms [110]. In 1992, Dorigo completed his PhD thesis, in which he describes his innovative work on ant colony optimization [69]. In 1993, the first algorithm based on bee colonies

* Corresponding author.
E-mail address: siarry@u-pec.fr (P. Siarry).

0020-0255/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2013.02.041

was proposed by Walker et al. [277]. Another significant progress is the development of the particle swarm optimization by Kennedy and Eberhart in 1995 [145]. The same year, Hansen and Ostermeier proposed CMA-ES [121]. In 1996, Mühlenbein and Paaß proposed the estimation of distribution algorithm [190]. In 1997, Storn and Price proposed differential evolution [253]. In 2002, Passino introduced an optimization algorithm based on bacterial foraging [200]. Then, Simon proposed a biogeography-based optimization algorithm in 2008 [247].

The considerable development of metaheuristics can be explained by the significant increase in the processing power of the computers, and by the development of massively parallel architectures. These hardware improvements relativize the CPU time–costly nature of metaheuristics.

A metaheuristic will be successful on a given optimization problem if it can provide a balance between the exploration (diversification) and the exploitation (intensification). Exploitation is needed to identify parts of the search space with high quality solutions. Exploitation is important to intensify the search in some promising areas of the accumulated search experience. The main differences between the existing metaheuristics concern the particular way in which they try to achieve this balance [28]. Many classification criteria may be used for metaheuristics. This may be illustrated by considering the classification of metaheuristics in terms of their features with respect to different aspects concerning the search path they follow, the use of memory, the kind of neighborhood exploration used or the number of current solutions carried from one iteration to the next. For a more formal classification of metaheuristics we refer the reader to [28,258]. The metaheuristic classification, which differentiates between Single-Solution Based Metaheuristics and Population-Based Metaheuristics, is often taken to be a fundamental distinction in the literature. Roughly speaking, basic single-solution based metaheuristics are more exploitation oriented whereas basic population-based metaheuristics are more exploration oriented.

The purpose of this paper is to present a global overview of the main metaheuristics and their principles. That attempt of survey on metaheuristics is structured in the following way. Section 2 shortly presents the class of single-solution based metaheuristics, and the main algorithms that belong to this class, i.e. the simulated annealing method, the tabu search, the GRASP method, the variable neighborhood search, the guided local search, the iterated local search, and their variants. Section 3 describes the class of metaheuristics related to population-based metaheuristics, which manipulate a collection of solutions rather than a single solution at each stage. Section 3.1 describes the field of evolutionary computation and outlines the common search components of this family of algorithms (e.g., selection, variation, and replacement). In this subsection, the focus is on evolutionary algorithms such as genetic algorithms, evolution strategies, evolutionary programming, and genetic programming. Section 3.2 presents other evolutionary algorithms such as estimation of distribution algorithms, differential evolution, coevolutionary algorithms, cultural algorithms and the scatter search and path relinking. Section 3.3 contains an overview of a family of nature inspired algorithms related to Swarm Intelligence. The main algorithms belonging to this field are ant colonies, particle swarm optimization, bacterial foraging, bee colonies, artificial immune systems and biogeography-based optimization. Finally, a discussion on the current research status and most promising paths of future research is presented in Section 4.

2. Single-solution based metaheuristics

In this section, we outline single-solution based metaheuristics, also called *trajectory methods*. Unlike population-based metaheuristics, they start with a single initial solution and move away from it, describing a trajectory in the search space. Some of them can be seen as "intelligent" extensions of local search algorithms. Trajectory methods mainly encompass the simulated annealing method, the tabu search, the GRASP method, the variable neighborhood search, the guided local search, the iterated local search, and their variants.

2.1. Simulated annealing

The origins of the Simulated Annealing method (SA) are in statistical mechanics (Metropolis algorithm [179]). It was first proposed by Kirkpatrick et al. [150], and independently by Cerny [42]. SA is inspired by the annealing technique used by the metallurgists to obtain a "well ordered" solid state of minimal energy (while avoiding the "metastable" structures, characteristic of the local minima of energy). This technique consists in carrying a material at high temperature, then in lowering this temperature slowly.

SA transposes the process of the annealing to the solution of an optimization problem: the objective function of the problem, similar to the energy of a material, is then minimized, by introducing a fictitious temperature T, which is a simple controllable parameter of the algorithm.

The algorithm starts by generating an initial solution (either randomly or constructed using an heuristic) and by initializing the temperature parameter T. Then, at each iteration, a solution s' is randomly selected in the neighborhood $N(s)$ of the current solution s. The solution s' is accepted as new current solution depending on T and on the values of the objective function for s' and s, denoted by $f(s')$ and $f(s)$, respectively. If $f(s') \leqslant f(s)$, then s' is accepted and it replaces s. On the other hand, if $f(s') > f(s)$, s' can also be accepted, with a probability $p(T, f(s'), f(s)) = \exp\left(-\frac{f(s')-f(s)}{T}\right)$. The temperature T is decreased during the search process, thus at the beginning of the search, the probability of accepting deteriorating moves is high and it gradually decreases. The high level SA algorithm is presented in Fig. 1.

SA

```
Choose, at random, an initial solution s for the system to be optimized
Initialize the temperature T
while the stopping criterion is not satisfied do
    repeat
        Randomly select s' ∈ N(s)
        if f(s') ≤ f(s) then
            s ← s'
        else
            s ← s' with a probability p(T, f(s'), f(s))
        end
    until the "thermodynamic equilibrium" of the system is reached
    Decrease T
end
return the best solution met
```

Fig. 1. Algorithm for the simulated annealing method.

One can notice that the algorithm can converge to a solution s, even if a better solution s' is met during the search process. Then, a basic improvement of SA consists in saving the best solution met during the search process.

SA has been successfully applied to several discrete or continuous optimization problems, though it has been found too greedy or unable to solve some combinatorial problems. The adaptation of SA to continuous optimization problems has been particularly studied [58]. A wide bibliography can be found in [57,90,152,157,255,259].

Several variants of SA have been proposed in the literature. Three of them are described below.

2.1.1. Microcanonic annealing

The principle of Microcanonic Annealing (MA) is similar to that of SA, and MA can be considered as a variant of SA. The main difference is that instead of using a Metropolis algorithm, MA uses the Creutz algorithm [59], known as microcanonical Monte Carlo simulation or "demon" algorithm. The Creutz algorithm allows reaching the thermodynamic equilibrium in an isolated system, i.e. a system where the total energy, which is the sum of the potential energy and the kinetic energy, remains constant ($E_{total} = E_p + E_c$).

For an optimization problem, the potential energy E_p can be considered as the objective function, to be minimized. The kinetic energy E_c is used in a similar manner to the temperature in simulated annealing; it is forced to remain positive. The algorithm accepts all disturbances which cause moves towards the lower energy states, by adding $-\Delta E$ (lost potential energy) to the kinetic energy E_c. The moves towards higher energy states are only accepted when $\Delta E < E_c$, and the energy acquired in the form of potential energy is cut off from the kinetic energy. Thus, the total energy remains constant. The MA algorithm is presented in Fig. 2.

At each energy stage, the "thermodynamic equilibrium" is reached as soon as the ratio $r_{eq} = \frac{\langle E_c \rangle}{\sigma(E_c)}$ of the average kinetic energy observed to the standard deviation of the distribution of E_c is "close" to 1.

Eq. (1) involving the kinetic energy and the temperature establishes a link between SA and MA, where k_B denotes the Boltzmann constant.

$$k_B\, T = \langle E_c \rangle \tag{1}$$

MA has several advantages compared to simulated annealing. It neither requires the transcendent functions like *exp* to be evaluated, nor any random number to be drawn for the acceptance or the rejection of a solution. Their computations can be indeed time costly. A relatively recent work shows the successful application of MA, hybridized with the Nelder-Mead simplex method [193], in microscopic image processing [191].

2.1.2. Threshold accepting method

Another variant of SA is the Threshold Accepting method (TA) [77]. The principal difference between TA and SA lies in the criterion for acceptance of the candidate solutions: on the one hand, SA accepts a solution that causes deterioration of the objective function f only with a certain probability; on the other hand, TA accepts this solution if the degradation of f does not exceed a progressively decreasing threshold T. The TA algorithm is presented in Fig. 3.

The TA method compares favorably with simulated annealing for combinatorial optimization problems, like the traveling salesman problem [76]. An adaptation of TA to continuous optimization can be carried out similarly to SA.

MA

```
Choose, at random, an initial solution s for the system to be optimized
Initialize the kinetic energy E_c
repeat
    repeat
        Randomly select s' ∈ N(s)
        Calculate ΔE = f(s') − f(s)
        if ΔE < E_c then
            s ← s'
            E_c ← E_c − ΔE
        end
    until the "thermodynamic equilibrium" of the system is reached
    Decrease E_c
until E_c is close to 0
return the best solution met
```

Fig. 2. Algorithm for the microcanonic annealing method.

TA

```
Choose, at random, an initial solution s in the search space
Initialize the threshold T
repeat
    repeat
        Randomly select s' ∈ N(s)
        Calculate Δf = f(s') − f(s)
        if Δf < T then
            s ← s'
        end
    until the best solution met is not improved for a certain duration, or a given number of iterations is reached
    Decrease T
until T is close to 0
return the best solution met
```

Fig. 3. Algorithm for the threshold accepting method.

2.1.3. Noising method

The Noising Method (NM) was proposed by Charon and Hudry [45]. Initially proposed for the *clique partitioning problem* in a graph, it has been shown to be successful for many combinatorial optimization problems. It uses a local search algorithm, i.e. an algorithm which, starting from an initial solution, carries out iterative improvements until obtaining a local optimum. The basic idea of NM is as follows. Rather than taking the genuine data of an optimization problem directly into account, the data are "perturbed", i.e. the values taken by the objective function are modified in a certain way. Then, the local search algorithm is applied to the perturbed function. At each iteration of NM, the amplitude of the noising of the objective function decreases until it is zero. The reason behind the addition of noise is to be able to escape any possible local optimum of the objective function. In NM, a noise is a value taken by a random variable following a given probability distribution (e.g. uniform or Gaussian law). The algorithm for NM is presented in Fig. 4.

The authors proposed and analysed different ways to add noise [46]. They showed that, according to the noising carried out, NM can be made identical with SA, or with TA, described above. Thus, NM represents a generalization of SA and TA. They also published a survey in [47], and recently, they proposed a way to design NM that can tune its parameters itself [48].

2.2. Tabu search

Tabu Search (TS) was formalized in 1986 by Glover [104]. TS was designed to manage an embedded local search algorithm. It explicitly uses the history of the search, both to escape from local minima and to implement an explorative strategy. Its main characteristic is indeed based on the use of mechanisms inspired by the human memory. It takes, from this point of view, a path opposite to that of SA, which does not use memory, and thus is unable to learn from the past.

```
NM
Choose, at random, an initial solution in the search space
Initialize the amplitude of noise
repeat
    Add noise to the objective function
    Apply local search to the "perturbed" objective function, starting from the current solution
    Decrease the amplitude of noise
until the amplitude of noise is zero
return the best solution met
```

Fig. 4. Algorithm for the noising method.

```
TS
Choose, at random, an initial solution s in the search space
TabuList ← ∅
while the stopping criterion is not satisfied do
    Select the best solution s' ∈ N(s) \ TabuList
    s ← s'
    Update TabuList
end
return the best solution met
```

Fig. 5. Algorithm for the simple tabu search method.

Various types of memory structures are commonly used to remember specific properties of the trajectory through the search space that the algorithm has undertaken. A *tabu list* (from which the name of the metaheuristic framework derives) records the last encountered solutions (or some attributes of them) and forbids these solutions (or solutions containing one of these attributes) from being visited again, as long as they are in the list. This list can be viewed as short-term memory, that records information on recently visited solutions. Its use prevents from returning to recently visited solutions, therefore it prevents from endless cycling and forces the search to accept even deteriorating moves. A simple TS algorithm is presented in Fig. 5.

The length of the tabu list controls the memory of the search process. If the length of the list is low, the search will concentrate on small areas of the search space. On the opposite, a high length forces the search process to explore larger regions, because it forbids revisiting a higher number of solutions. This length can be varied during the search, leading to more robust algorithms, like the Reactive Tabu Search algorithm [19].

Additional intermediate-term memory structures can be introduced to bias moves towards promising areas of the search space (intensification), as well as long-term memory structures to encourage broader exploration of the search space (diversification).

The addition of intermediate-term memory structures, called *aspiration criteria*, can greatly improve the search process. Indeed, the use of a tabu list can prevent attractive moves, even if there is no risk of cycling, or they may lead to an overall stagnation of the search process. For example, a move which leads to a solution better than all those visited by the search in the preceding iterations does not have any reason to be prohibited. Then, the aspiration criteria, that are a set of rules, are used to override tabu restrictions, i.e. if a move is forbidden by the tabu list, then the aspiration criteria, if satisfied, can allow this move.

A frequency memory can also be used as a type of long-term memory. This memory structure records how often certain attributes have been encountered in solutions on the search trajectory, which allows the search to avoid visiting solutions that present the most often encountered attributes or to visit solutions with attributes rarely encountered.

An extensive description of TS and its concepts can be found in [107]. Good reviews of the method are provided in [100,101]. TS was designed for, and has predominately been applied to combinatorial optimization problems. However, adaptations of TS to continuous optimization problems have been proposed [105,60,49].

2.3. GRASP method

GRASP, for *Greedy Randomized Adaptive Search Procedure*, is a memory-less multi-start metaheuristic for combinatorial optimization problems, proposed by Feo and Resende in [84,85]. Each iteration of the GRASP algorithm consists of two

GRASP

1 **repeat**
2 | Build a feasible solution using a randomized greedy heuristic
3 | Apply a local search starting from the built solution
4 **until** the stopping criterion is satisfied
5 **return** the best solution met

Fig. 6. Template for the GRASP algorithm.

steps: construction and local search. The construction step of GRASP is similar to the *semi-greedy heuristic* proposed independently by Hart and Shogan [129]. The construction step builds a feasible solution using a randomized greedy heuristic. In the second step, this solution is used as the initial solution of a local search procedure. After a given number of iterations, the GRASP algorithm terminates and the best solution found is returned. A template for the GRASP algorithm is presented in Fig. 6.

In the greedy heuristic, a candidate solution is built iteratively, i.e. at each iteration, an element is incorporated into a partial solution, until a complete solution is built. It means that, for a given problem, one has to define a solution as a set of elements. At each iteration of the heuristic, the list of candidate elements is formed by all the elements that can be included in the partial solution, without destroying feasibility. This list is ordered with respect to a greedy function, that measures the benefit of selecting each element. Then, the element to be added to the partial solution is randomly chosen among the best candidates in this list. The list of the best candidates is called the *restricted candidate list* (RCL). This random selection of an element in the RCL represents the probabilistic aspect of GRASP. The RCL list can be limited either by the number of elements (cardinality-based) or by their quality (value-based). In the first case, the RCL list consists of the p best candidate elements, where p is a parameter of the algorithm. In the second case, it consists of the candidate elements having an *incremental cost* (the value of the greedy function) greater or equal to $c^{min} + \alpha(c^{max} - c^{min})$, where α is a parameter of the algorithm, and c^{min} and c^{max} are the values of the best and worst elements, respectively. This is the most used strategy [221], and α is the main parameter of GRASP, where $\alpha \in [0,1]$. Indeed, this parameter defines the compromise between intensification and diversification.

The performance of GRASP is very sensitive to the α parameter, and many strategies have been proposed to fit it [258] (initialized to a constant value, dynamically changed according to some probability distribution, or automatically adapted during the search process). A self-tuning of α is performed in *reactive GRASP*, where the value of α is periodically updated according to the quality of the obtained solutions [214].

Festa and Resende surveyed the algorithmic aspects of GRASP [86], and its application to combinatorial optimization problems [87]. A good bibliography is provided also in [222]. GRASP can be hybridized in different ways, for instance by replacing the local search with another metaheuristic such as tabu search, simulated annealing, variable neighborhood search, iterated local search, among others [220,271,235]. It is also often combined with a path-relinking strategy [88]. Adaptations to continuous optimization problems have also been proposed [134].

2.4. Variable neighborhood search

Variable Neighborhood Search (VNS) is a metaheuristic proposed by Hansen and Mladenovic [184,186]. Its strategy consists in the exploration of dynamically changing neighborhoods for a given solution. At the initialization step, a set of neighborhood structures has to be defined. These neighborhoods can be arbitrarily chosen, but often a sequence $N_1, N_2, \ldots, N_{n_{max}}$ of neighborhoods with increasing cardinality is defined. In principle they could be included one in the other ($N_1 \in N_2 \in \ldots \in N_{n_{max}}$). However, such a sequence may produce an inefficient search, because a large number of solutions can be revisited [33]. Then an initial solution is generated, and the main cycle of VNS begins. This cycle consists of three steps: *shaking*, *local search* and *move*. In the shaking step, a solution s' is randomly selected in the nth neighborhood of the current solution s. Then, s' is used as the initial solution of a local search procedure, to generate the solution s''. The local search can use any neighborhood structure and is not restricted to the set N_n, $n = 1, \ldots, n_{max}$. At the end of the local search process, if s'' is better than s, then s'' replaces s and the cycle starts again with $n = 1$. Otherwise, the algorithm moves to the next neighborhood $n + 1$ and a new shaking phase starts using this neighborhood. The VNS algorithm is presented in Fig. 7.

This algorithm is efficient if the neighborhoods used are complementary, i.e. if a local optimum for a neighborhood N_i is not a local optimum for a neighborhood N_j. VNS is based on the variable neighborhood descent (VND), which is a deterministic version of VNS [258] described in Fig. 8. A more general VNS algorithm (GVNS), where VND is used as the local search procedure of VNS, has led to many successful applications [122]. Recent surveys of VNS and its extensions are available in [122,123]. Adaptations to continuous optimization problems have been proposed in [164,185,37]. Hybridization of VNS with other metaheuristics, such as GRASP, is also common [271,235]. The use of more than one neighborhood structure is not restricted to algorithms labeled VNS [250]. In Reactive Search [18], a sophisticated adaptation of the neighborhood is performed, instead of cycling over a predefined set of neighborhoods.

```
VNS
Select a set of neighborhood structures N_n, n = 1, ..., n_max
Choose, at random, an initial solution s in the search space
while the stopping criterion is not satisfied do
    n ← 1
    while n < n_max do
        Shaking: select a random solution s' in the n^th neighborhood N_n(s) of s
        Apply a local search starting from s' to get a solution s''
        if s'' is better than s then
            s ← s''
            n ← 1
        else
            n ← n + 1
        end
    end
end
return the best solution met
```

Fig. 7. Template for the variable neighborhood search algorithm.

```
VND
Select a set of neighborhood structures N_n, n = 1, ..., n_max
Choose, at random, an initial solution s in the search space
n ← 1
while n < n_max do
    Select the best solution s' in the n^th neighborhood N_n(s) of s
    if s' is better than s then
        s ← s'
        n ← 1
    else
        n ← n + 1
    end
end
return the best solution met
```

Fig. 8. Template for the variable neighborhood descent algorithm.

2.5. Guided local search

As tabu search, Guided Local Search (GLS) [272,274] makes use of a memory. In GLS, this memory is called an *augmented objective function*. Indeed, GLS dynamically changes the objective function optimized by a local search, according to the found local optima. First, a set of features $ft_n, n = 1, \ldots, n_{max}$ has to be defined. Each feature defines a characteristic of a solution regarding the optimization problem to solve. Then, a cost c_i and a penalty value p_i are associated with each feature. For instance, in the traveling salesman problem, a feature ft_i can be the presence of an edge from a city A to a city B in the solution, and the corresponding cost c_i can be the distance, or the travel time, between these two cities. The penalties are initialized to 0 and updated when the local search reaches a local optimum. Given an objective function f and a solution s, GLS defines the augmented objective function f' as follows:

$$f'(s) = f(s) + \lambda \sum_{i=1}^{n_{max}} p_i I_i(s) \tag{2}$$

where λ is a parameter of the algorithm, and $I_i(s)$ is an indicator function that determines whether s exhibits the feature ft_i:

$$I_i(s) = \begin{cases} 1 & \text{if } s \text{ exhibits the feature } ft_i \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Each time a local optimum is found by the local search, GLS intends to penalize the most "unfavorable features" of this local optimum, i.e. the features having a high cost. This way, solutions exhibiting other features become more attractive, and the search can escape from the local optimum. When a local optimum s^* is reached, the utility u_i of penalizing a feature ft_i is calculated as follows:

$$u(s^*) = I_i(s^*)\frac{c_i}{1+p_i} \tag{4}$$

The greater the cost c_i of this feature, the greater the utility to penalize it is. Besides, the more times it has been penalized (the greater p_i), the lower the utility of penalizing it again is. The feature having the greatest utility value is penalized: its penalty value p_i is increased by *1*. In the augmented objective function, the scaling of the penality is adjusted by λ. Authors suggest that the performance of GLS is not very sensitive to the value of λ [183]. Large values of λ encourage diversification, while small values intensify the search around the local optimum [258]. GLS algorithm is summarized in Fig. 9.

Sitting on top of a local search algorithm, the adaptation of GLS to continuous optimization is straightforward [273]. Extensions to population based metaheuristics have been proposed [160,286,257]. Mills et al. proposed an extended guided local search algorithm (EGLS), adding *aspiration criteria* and random moves to GLS [182,183]. A recent survey on GLS and its applications is available in [276,275].

2.6. Iterated local search

The definition and framework of Iterated Local Search (ILS) are given by Stützle in his PhD dissertation [254]. Stützle does not take credit for the approach, and instead highlights specific instances of ILS from the literature, such as *iterated descent* [20], *large-step Markov chains* [176], *iterated Lin-Kernighan* [139], *chained local optimization* [175], as well as [21] that introduces the principle, and [140] that summarizes it (list taken from [168,36]).

ILS is a metaheuristic based on a simple idea: instead of repeatedly applying a local search procedure to randomly generated starting solutions, ILS generates the starting solution for the next iteration by perturbing the local optimum found at

GLS

1 Select a set of features $ft_n, n = 1, ..., n_{max}$
2 Choose, at random, an initial solution s in the search space
3 Initialize the penalties p_i to 0
4 **repeat**
5 | Apply a local search starting from s to get a solution s^*, using the augmented objective function given in (2)
6 | **for each** feature ft_i of s^* **do**
7 | | Compute the utility u_i of ft_i as in (4)
8 | **end**
9 | $j \leftarrow \arg\max_{i=1}^{n_{max}} u_i$
10 | $p_j \leftarrow p_j + 1$
11 **until** the stopping criterion is satisfied
12 **return** the best solution met

Fig. 9. Template for the guided local search algorithm.

ILS

1 Choose, at random, an initial solution s in the search space
2 Apply a local search starting from s to get a solution s^*
3 **repeat**
4 | Perturb s^* to get a solution p
5 | Apply a local search starting from p to get a solution p^*
6 | **if** the acceptance criterion is satisfied **then**
7 | | $s^* \leftarrow p^*$
8 | **end**
9 **until** the stopping criterion is satisfied
10 **return** the best solution met

Fig. 10. Template for the iterated local search algorithm.

the current iteration. This is done in the expectation that the perturbation mechanism provides a solution located in the basin of attraction of a better local optimum. The perturbation mechanism is a key feature of ILS: on the one hand, a too weak perturbation may not be sufficient to escape from the basin of attraction of the current local optimum; on the other hand, a too strong perturbation would make the algorithm similar to a multistart local search with randomly generated starting solutions. ILS algorithm is summarized in Fig. 10, where the *acceptance criterion* defines the conditions that the new local optimum p^* has to satisfy in order to replace the current one s^*.

The acceptance criterion, combined with the perturbation mechanism, enables controlling the trade-off between intensification and diversification. For instance, an extreme acceptance criterion in terms of intensification is to accept only improving solutions. Another extreme criterion in terms of diversification is to accept any solution, without regard to its quality. Many acceptance criteria that balance the two goals may be applied [258].

A recent review of ILS, its extensions and its applications is available in [168].

3. Population-based metaheuristics

Population-based metaheuristics deal with a set (i.e. a population) of solutions rather than with a single solution. The most studied population-based methods are related to Evolutionary Computation (EC) and Swarm Intelligence (SI). EC algorithms are inspired by Darwin's evolutionary theory, where a population of individuals is modified through recombination and mutation operators. In SI, the idea is to produce computational intelligence by exploiting simple analogs of social interaction, rather than purely individual cognitive abilities.

3.1. Evolutionary computation

Evolutionary Computation (EC) is the general term for several optimization algorithms that are inspired by the Darwinian principles of nature's capability to evolve living beings well adapted to their environment. Usually found grouped under the term of EC algorithms (also called Evolutionary Algorithms (EAs)), are the domains of genetic algorithms [135], evolution strategies [217], evolutionary programming [95], and genetic programming [154]. Despite the differences between these techniques, which will be shown later, they all share a commun underlying idea of simulating the evolution of individual structures via processes of selection, recombination, and mutation reproduction, thereby producing better solutions.

A generic form of a basic EA is shown in Fig. 11. This form will serve as a template for algorithms that will be discussed throughout this section.

Every iteration of the algorithm corresponds to a *generation*, where a *population* of candidate solutions to a given optimization problem, called *individuals*, is capable of reproducing and is subject to genetic variations followed by the environmental pressure that causes natural selection (survival of the fittest). New solutions are created by applying *recombination*, that combines two or more selected individuals (the so-called *parents*) to produce one or more new individuals (the *children* or *offspring*), and *mutation*, that allows the appearance of new traits in the offspring to promote diversity. The *fitness* (how good the solutions are) of the resulting solutions is evaluated and a suitable selection strategy is then applied to determine which solutions will be maintained into the next generation. As a termination condition, a predefined number of generations (or function evaluations) of simulated evolutionary process is usually used, or some more complex stopping criteria can be applied.

Over the years, there have been many overviews and surveys about EAs. The readers interested in the history are referred to [13,10]. EAs have been widely applied with a good measure of success to combinatorial optimization problems [33,26], constrained optimization problems [55], Data Mining and Knowledge Discovery [97], etc. Multi-Objective Evolutionary Algorithms (MOEAs) are one of the current trends in developing EAs. An excellent overview of current issues, algorithms, and existing systems in this area is presented in [54]. Parallel EAs have also deserved interest in the recent past (a good review

EC

1 *Initialize* the population with random individuals
2 *Evaluate* each individual
3 **repeat**
4 *Select* parents
5 *Recombine* pairs of parents
6 *Mutate* the resulting offspring
7 *Evaluate* new individuals
8 *Select* individuals for the next generation
9 **until** a termination condition is satisfied

Fig. 11. Evolutionary computation algorithm.

can be found in [5]). Another overview paper over self-adaptive methods in EAs is given in [180]. They are called self-adaptive, because these algorithms control the setting of their parameters themselves, embedding them into an individual's genome and evolving them. Some other topics covered by the literature include the use of hybrid EAs, that combine local search or some other heuristic search methods [32].

3.1.1. Genetic algorithm

The Genetic Algorithm (GA) is arguably the most well-known and mostly used evolutionary computation technique. It was originally developed in the early 1970s at the University of Michigan by John Holland and his students, whose research interests were devoted to the study of adaptive systems [135]. The basic GA is very generic, and there are many aspects that can be implemented differently according to the problem: representation of solution (*chromosomes*), selection strategy, type of crossover (the recombination operator of GAs) and mutation operators, etc. The most common representation of the chromosomes applied in GAs is a fixed-length *binary* string. Simple bit manipulation operations allow the implementation of *crossover* and *mutation* operations. These genetic operators form the essential part of the GA as a problem-solving strategy. Emphasis is mainly concentrated on crossover as the main variation operator, that combines multiple (usually two) individuals that have been selected together by exchanging some of their parts. There are various strategies to do this, e.g. *n-point* and *uniform crossover*. An exogenous parameter p_c (*crossover rate*) indicates the probability per individual to undergo crossover. Typical values for p_c are in the range [0.6,1.0] [13]. Individuals for producing offspring are chosen using a selection strategy after evaluating the fitness value of each individual in the selection pool. Some of the popular selection schemes are *roulette-wheel selection*, *tournament selection*, *ranking selection*, etc. A comparison of selection schemes used in GAs is given in [111,30]. After crossover, individuals are subjected to mutation. Mutation introduces some randomness into the search to prevent the optimization process from getting trapped into local optima. It is usually considered as a secondary genetic operator that performs a slight perturbation to the resulting solutions with some low probability p_m. Typically, the mutation rate is applied with less than 1% probability, but the appropriate value of the mutation rate for a given optimization problem is an open research issue. The *replacement* (*survivor selection*) uses the fitness value to identify the individuals to maintain as parents for successive generations and is responsible to assure the survival of the fittest individuals. Interested readers may consult the book by Goldberg [110] for more detailed background information on GAs.

Since then, many variants of GAs have been developed and applied to a wide range of optimization problems. Overviews concerning current issues on GAs can be found in [22,23], [249] for hybrid GAs, [151] for multi-objective optimization and [6] for Parallel GAs, among others. Indexed bibliographies of GAs have been compiled by Jarmo T. Alander in various application areas, like in robotics, Software Engineering, Optics and Image Processing, etc. Versions of these bibliographies are available via anonymous `ftp` or `www` from the following site: `ftp.uwasa.fi/cs/report94-1`.

3.1.2. Evolution Strategy

Similar to GA, Evolution Strategy (ES) imitates the principles of natural evolution as a method to solve optimization problems. It was introduced in the 60ies by Rechenberg [216,217] and further developed by Schwefel. The first ES algorithm, used in the field of experimental parameter optimization, was a simple mutation-selection scheme called *two membered ES*. Such ES is based upon a population consisting of a single parent which produces, by means of normally (Gaussian) distributed mutation, a single descendant. The selection operator then determines the fitter individual to become the parent of the next generation.

To introduce the concept of population, which has not really been used so far, Rechenberg proposed the *multimembered ES*, where $\mu > 1$ parents can participate in the generation of one offspring individual. This has been denoted by $(\mu + 1) - \text{ES}$. With the introduction of more than one parent, an additional recombination operator is possible. Two of the μ parents are chosen at random and recombined to give life to an offspring, which also underlies mutation. The selection resembles "extinction of the worst", may it be the offspring or one of the parents, thus keeping constant the population size. Schwefel [237] introduced two further versions of *multimembered ES*, i.e. $(\mu + \lambda) - \text{ES}$ and $(\mu,\lambda) - \text{ES}$. The first case indicates that μ parents create $\lambda \geqslant 1$ descendants by means of recombination and mutation, and, to keep the population size constant, the λ worst out of all $\mu + \lambda$ individuals are discarded. For a $(\mu,\lambda) - \text{ES}$, with $\lambda > \mu$, the μ best individuals of the λ offspring become the parents of the next population, whereas their parents are deleted, no matter how good or bad their fitness was compared to that of the new generation's individuals. Two other well-known ES versions are known as $(\mu/\rho + \lambda) - \text{ES}$ and $(\mu/\rho,\lambda) - \text{ES}$. The additional parameter ρ refers to the number of parents involved in the procreation of one offspring.

The mutation in ES is realized through normally distributed numbers with zero mean and standard deviation σ, which can be interpreted as the *mutation step size*. It is easy to imagine that the parameters of the normal distribution play an essential role for the performance of the search algorithm. The simplest method to specify the mutation mechanism is to keep σ constant over time. Another approach consists in dynamically adjusting σ by assigning different values depending on the number of generations or by incorporating feedback from the search process. Various methods to control the mutation parameter have been developed. Among these there are for example Rechenberg's 1/5 success rule[1] [217], the

[1] The 1/5 rule in $(1 + 1) - \text{ES}$ states that: *the ratio of successful mutations to all mutations should be 1/5. If it is greater than 1/5, increase the variance; if it is less, decrease the mutation variance.*

σ-self-adaptation (σSA)[2] [217], the meta-ES (mES)[3] [239], a hierarchically organized population based ES involving isolation periods [132], the mutative self adaptation [238], the machine learning approaches [240], or the cumulative pathlength control [197].

Adaptivity is not limited to a single parameter, like the step-size. More recently, a surprisingly effective method, called the *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES), was introduced by Hansen, Ostermeier, and Gawelczyk [121] and further developed in [120]. The CMA-ES is currently the most widely used, and it turns out to be a particularly reliable and highly competitive EA for local optimization [120] and also for global optimization [119]. In the "Special Session on Real-Parameter Optimization" held at the CEC 2005 Congress, the CMA-ES algorithm obtained the best results among all the evaluated techniques on a benchmark of 25 continuous functions [9]. The performances of the CMA-ES algorithm are also compared to those of other approaches in the Workshop on Black-Box Optimization Benchmarking BBOB'2009 and on the test functions of the BBOB'2010.

In recent years, a fair amount of theoretical investigation has contributed substantially to the understanding of the evolutionary search strategies on a variety of problem classes. A number of review papers and text books exist with such details to which the reader is referred (see [11,13,25,156]).

3.1.3. Evolutionary programming

Evolutionary Programming (EP) was first presented in the 1960s by L.J. Fogel as an evolutionary approach to artificial intelligence [95]. Later, in the early 1990s, EP was reintroduced by D. Fogel to solve more general tasks including prediction problems, numerical and combinatorial optimization, and machine learning [91,92].

The representations used in EP are typically tailored to the problem domain. In real-valued vector optimization, the coding will be taken naturally as a string of real values. The initial population is selected at random with respect to a density function and is scored with respect to the given objective. In contrast to the GAs, the conventional EP does not rely on any kind of recombination. The mutation is the only operator used to generate new offspring. It is implemented by adding a random number of certain distributions to the parent. In the case of standard EP, the normally distributed random mutation is applied. However, other mutation schemes have been proposed. D. Fogel [93] developed an extension of the standard EP, called *meta-EP*, that self-adapts the standard deviations (or equivalently the variances). The *R-meta-EP* algorithm [94] incorporates the self-adaptation of covariance matrices in addition to standard deviations. Yao and Liu [284] substituted the normal distribution of the *meta-EP* operator with a Cauchy-distribution in their new algorithm, called *fast evolutionary programming* (FEP). In [162], Lee and Yao proposed to use a Levy-distribution for higher variations and a greater diversity. In Yao's *Improved Fast Evolutionary Programming* algorithm (IFEP) [285], two offspring are created from each parent, one using a Gaussian distribution, and the other using the Cauchy distribution. The parent selection mechanism is deterministic. The survivor selection process (replacement) is probabilistic and is based on a stochastic tournament selection. The framework of EP is less used than the other families of EAs, due to its similarity with ES, as it turned out in [12].

3.1.4. Genetic programming

The Genetic Programming (GP) became a popular search technique in the early 1990s due to the work by Koza [154]. It is an automated method for creating a working computer program from a high-level problem statement of "*what needs to be done*".

GP adopts a similar search strategy as a GA, but uses a program representation and special operators. In GP, the individual population members are not fixed-length strings as used in GAs, they are computer programs that, when executed, are the candidate solutions to the problem at hand. These programs are usually expressed as *syntax trees* rather than as lines of code, which provides a flexible way of describing them in LISP language, as originally used by J. Koza. The variables and constants in the program, called *terminals* in GP, are leaves of the tree, while the arithmetic operations are internal nodes (typically called *functions*). The terminal and function sets form the alphabets of the programs to be made.

GP starts with an initial population of randomly generated computer programs composed of functions and terminals appropriate to the problem domain. There are many ways to generate the initial population resulting in initial random trees of different sizes and shapes. Two of the basic ways, used in most GP systems are called *full* and *grow* methods. The *full* method creates trees for which the length of every nonbacktracking path between an endpoint and the root is equal to the specified maximum depth.[4] The *grow* method involves growing trees that are variably shaped. The length of a path between an endpoint and the root is no greater than the specified maximum depth. A widely used combination of the two methods, known as *Ramped half-and-half* [154], involves creating an equal number of trees using a depth parameter that ranges between 2 and the maximum specified depth. While these methods are easy to implement and use, they often make it difficult to control the statistical distributions of important properties such as the sizes and shapes of the generated trees [280]. Other initialization mechanisms, however, have been developed to create different distributions of initial trees, where the general consensus is that a more uniform and random distribution is better for the evolutionary process [170].

[2] Instead of changing σ by an exogenous heuristic in a deterministic manner, Schwefel completely viewed σ as a part of genetic information of an individual, which can be interpreted as *self-adaptation of step sizes*. Consequently, it is subject to recombination and mutation as well.

[3] σSA and mES do not exclude each other. A mES may perform SA and a SA can include a lifetime mechanism allowing a variable lifespan of certain individuals.

[4] The depth of a tree is defined as the length of the longest nonbacktracking path from the root to an endpoint.

The population of programs is then progressively evolved over a series of generations, using the principles of Darwinian natural selection and biologically inspired operations, including crossover and mutation, which are specialized to act on computer programs. To create the next population of individuals, computer programs are probabilistically selected, in proportion to fitness, from the current population of programs. That is, better individuals are more likely to have more child programs than inferior individuals. The most commonly employed method for selecting individuals in GP is *tournament selection*, followed by *fitness-proportionate selection*, but any standard EA selection mechanism can be used [209]. Recombination is usually implemented as *subtree crossover* between two parents. The resulting offspring are composed of subtrees from their parents that may be of different sizes and in different positions in their programs. Other forms of crossover have been defined and used, such as *one-point crossover*, *context-preserving crossover*, *size-fair crossover* and *uniform crossover* [209]. Mutation is another important feature of GP. The most commonly used form of mutation is *subtree mutation*, which randomly selects a mutation point in a tree and substitutes the subtree rooted there with a randomly generated subtree. Other forms of mutation include *single-node mutations* and various forms of code-editing to remove unnecessary code from trees have been proposed in the literature [209,206]. Also, often, in addition to crossover and mutation, an operation which simply copies selected individuals in the next generation is used. This operation, called *reproduction*, is typically applied only to produce a fraction of the new generation [280]. The *replacement* phase concerns the survivor selection of both parent and offspring populations. There are two alternatives for implementing this step: the *generational* approach, where the offspring population will replace systematically the parent population and the *steady-state* approach, where the parent population is maintained and some of its individuals are replaced by new individuals according to some rules. Advanced GP issues concern developing *automatically defined functions* and specialized operators, such as *permutation*, *editing*, or *encapsulation* [155].

The theoretical foundations of GP as well as a review of many real-world applications and important extensions of GP are given in [209,280,177]. Contemporary GPs are widely used in machine learning and data mining tasks, such as prediction and classification. There is also a great amount of work done on GP using probabilistic models. The interested reader should refer to [243] which is a review that includes directions for further research on this area.

3.2. Other evolutionary algorithms

Other models of evolutionary algorithms have been proposed in the literature. Among them, one can find estimation of distribution algorithms, differential evolution, coevolutionary algorithms, cultural algorithms and Scatter Search and Path Relinking.

3.2.1. Estimation of distribution algorithms

Estimation of Distribution Algorithms (EDAs), also referred to as Probabilistic Model-Building Genetic Algorithms (PMBGA), were introduced in the field of evolutionary computation, for the first time, by Mühlenbein and Paaß[190]. These algorithms are based on probabilistic models, where genetic recombination and mutation operators of GA are replaced by the following two steps: (1) estimate the probability distribution of selected individuals (promising solutions) and (2) generate new population by sampling this probability distribution. This leads the search towards promising areas of the space of solutions. The new solutions are then incorporated into the original population, replacing some of the old ones or all of them. The process is repeated until the termination criteria are met. The type of probabilistic models used by EDAs and the methods employed to learn them may vary according to the characteristics of the optimization problem.

Based on this general framework, several EDA approaches have been developed in the last years, where each approach learns a specific probabilistic model that conditions the behavior of the EDA from the point of view of complexity and performance. EDAs can be broadly divided into three classes, according to the complexity of the probabilistic models used to capture the interdependencies between the variables: starting with methods that assume total independency between problem variables (*univariate* EDAs), through the ones that take into account some pairwise interactions (*bivariate* EDAs), to the methods that can accurately model even a very complex problem structure with highly overlapping multivariate building blocks (*multivariate* EDAs) [159].

In all the approaches belonging to the first category, it is assumed that the n-dimensional joint probability distribution of solutions can be factored as a product of independent univariate probability distributions. Algorithms based on this principle work very well on linear problems where the variables are not mutually interacting [188]. It must be noted that, in difficult optimization problems, different dependency relations can appear between variables and, hence, considering all of them independent may provide a model that does not represent the problem accurately. Common *univariate* EDAs include Population Based Incremental Learning (PBIL)[14], Univariate Marginal Distribution Algorithm (UMDA) [188] and Compact Genetic Algorithm (cGA) [125].

In contrast to univariate EDAs, algorithms in the *bivariate* EDAs category consider dependencies between pairs of variables. In this case, it is enough to consider second-order statistics. Examples of such algorithms are Mutual Information Maximizing Input Clustering algorithm (MIMIC) [66], Combining Optimizers with Mutual Information Trees (COMIT) [15] and Bivariate Marginal Distribution Algorithm (BMDA) [205]. These algorithms reproduce and mix building blocks of order two very efficiently, and therefore they work very well on linear and quadratic problems. Nonetheless, capturing only some pair-wise interactions has still shown to be insufficient for solving problems with multivariate or highly overlapping building blocks. That is why *multivariate* EDAs algorithms have been proposed.

Algorithms belonging to this last category use statistics of order greater than two to factorize the probability distribution. In this way, multivariate interactions between problem variables can be expressed properly without any kind of initial restriction. The best known *multivariate* EDAs are Bayesian Optimization Algorithm (BOA) [202,203], Estimation of Bayesian Networks Algorithm (EBNA) [82], Factorized Distribution Algorithm (FDA) [189], Extended Compact Genetic Algorithm (EcGA) [124] and Polytree Approximation Distribution Algorithm (PADA) [251]. Those EDAs that look for multi-dependencies are capable of solving many hard problems accurately, and reliably with the sacrifice of computation time due to the complexity of the learning interactions among variables. Nonetheless, despite increased computational time, the number of evaluations of the optimized function is reduced significantly. That is why the overall time complexity is significantly reduced for large problems [204].

EDAs have been applied to a variety of problems in domains such as engineering, biomedical informatics, and robotics. A detailed overview of different EDA approaches in both discrete and continuous domains can be found in [159] and a recent survey was published in [130]. However, despite their successful application, there are a wide variety of open questions [236] regarding the behavior of this type of algorithms.

3.2.2. Differential evolution

Differential Evolution (DE) algorithm is one of the most popular algorithm for the continuous global optimization problems. It was proposed by Storn and Price in the 90's [253] in order to solve the Chebyshev polynomial fitting problem and has proven to be a very reliable optimization strategy for many different tasks.

Like any evolutionary algorithm, a population of candidate solutions for the optimization task to be solved is arbitrarily initialized. For each generation of the evolution process, new individuals are created by applying reproduction operators (crossover and mutation). The fitness of the resulting solutions is evaluated and each individual (*target individual*) of the population competes against a new individual (*trial individual*) to determine which one will be maintained into the next generation. The *trial individual* is created by recombining the target individual with another individual created by mutation (called *mutant individual*). Different variants of DE have been suggested by Price et al. [215] and are conventionally named *DE/ x/ y/ z*, where *DE* stands for Differential Evolution, *x* represents a string that denotes the base vector, i.e. the vector being perturbed, whether it is "*rand*" (a randomly selected population vector) or "*best*" (the best vector in the population with respect to fitness value), *y* is the number of difference vectors considered for perturbation of the base vector *x* and *z* denotes the crossover scheme, which may be *binomial* or *exponential*. The DE/rand/1/bin-variant, also known as the classical version of DE, is used later on for the description of the DE algorithm.

The mutation in DE is performed by calculating vector differences between other randomly selected individuals of the same population. There are several variants how to generate the *mutant individual*. The most frequently used mutation strategy (called *DE/rand/1/bin*) generates the trial vector $\overrightarrow{V}_{i,g}$ by adding only one weighted difference vector $F(\overrightarrow{X}_{r_2,g} - \overrightarrow{X}_{r_3,g})$ to a randomly selected base vector $\overrightarrow{X}_{r_1,g}$ to perturb it. Specifically, for each target vector $\overrightarrow{X}_{i,g}$, $i = 1,2,\ldots,N$, where g denotes the current generation and N the number of individuals in the population, a mutant vector is produced using the following formula:

$$\overrightarrow{V}_{i,g} = \overrightarrow{X}_{r_1,g} + F(\overrightarrow{X}_{r_2,g} - \overrightarrow{X}_{r_3,g}) \tag{5}$$

where the indexes r_1, r_2 and r_3 are randomly chosen over $[1,N]$ and should be mutually different from the running index i. F is a real constant scaling factor within the range $[0,1]$.

Based on the mutant vector, a trial vector $\overrightarrow{U}_{i,g}$ is constructed through a crossover operation which combines components from the ith population vector $\overrightarrow{X}_{i,g}$ and its corresponding mutant vector $\overrightarrow{V}_{i,g}$:

$$U_{i,j,g} = \begin{cases} V_{i,j,g} & \text{if } rand(0,1) \leqslant CR \text{ or } j = j_{rand} \\ X_{i,j,g} & \text{otherwise} \end{cases} \tag{6}$$

The crossover factor CR is randomly taken from the interval $[0,1]$ and presents the probability of creating parameters for trial vector from a mutant vector. Index j_{rand} is a randomly chosen integer within the range $[1,N]$. It is responsible for the trial vector containing at least one parameter from the mutant vector. $rand(0,1)$ is a uniform random number in range $[0,1]$. $j = 1, 2, \ldots, D$, where D is the number of parameters (dimension) of a single vector.

Finally, to decide whether or not it should become a member of generation $g + 1$, the trial vector $\overrightarrow{U}_{i,g}$ is compared to the target vector $\overrightarrow{X}_{i,g}$ using the fitness function evaluation:

$$\overrightarrow{X}_{i,g+1} = \begin{cases} \overrightarrow{U}_{i,g} & \text{if } f(\overrightarrow{U}_{i,g}) < f(\overrightarrow{X}_{i,g}) \\ \overrightarrow{X}_{i,g} & \text{otherwise} \end{cases} \tag{7}$$

The main advantage of the differential evolution consists in its fewer control parameters. It has only three input parameters controlling the search process, namely the population size N, the constant of differentiation F, which controls the amplification of the differential variation, and the crossover control parameter CR. In the original DE, the control parameters are kept fixed during the optimization process. It is not obvious to define *a priori* which parameter setting should be used as this task is problem specific. Therefore, some researchers (see for example [166,261,35]) have developed various strategies to make the setting of the parameters self-adaptive according to the learning experience.

DE is currently one of the most popular heuristics to solve single-objective optimization problems in continuous search spaces. Due to this success, its use has been extended to other types of problems, such as multi-objective optimization [181]. However, DE has certain flaws, like slow convergence and stagnation of population. Several modified versions of DE are available in literature for improving the performance of basic DE. One class of such algorithms includes hybridized versions, where DE is combined with some other algorithm to produce a new algorithm. For a more detailed description of many of the existing variants and major application areas of DE, readers should refer to [43,194,63].

3.2.3. Coevolutionary algorithms

When organisms that are ecologically intimate – for example, predators and prey, hosts and parasites, or insects and the flowers that they pollinate – influence each other's evolution, we say that *coevolution* is occurring. Biological coevolution encountered in many natural processes has been an inspiration for coevolutionary algorithms (CoEA), where two or more populations of individuals, each adapting to changes in the other, constantly interact and co-evolve simultaneously in contrast with traditional single population EAs.

Significant researching into the CoEAs began in the early 1990's with the seminal work of Hillis [133] on sorting networks. Contrary to conventional EAs, in which individuals are evaluated independently of one another through an absolute fitness measure, the individual fitness in CoEAs is *subjective*, in the sense that it is a function of its interactions with other individuals.

Many variants of CoEAs have been implemented since the beginning of 1990s. These variants fall into two categories: *competitive coevolution* and *cooperative coevolution*. In the case of *competitive* approaches, the different populations compete in solving the global problem and individuals are rewarded at the expense of those with which they interact. In the case of *cooperative* approaches, however, the various isolated populations are coevolved to cooperatively solve the problem; therefore individuals are rewarded when they work well with other individuals and punished when they perform poorly together.

Competitive coevolution is usually used to simulate the behavior of competing forces in nature, such as predators and prey where there is a strong evolutionary pressure for prey to defend themselves better, as future generations of predators develop better attacking strategies. Competitive coevolution can lead to an *arms race*, in which the two populations have opposing interests and the success of one population depends on the failure of the other. The idea is that continued minor adaptations in some individuals will force competitive adaptations in others, and these reciprocal forces will drive the algorithms to generate individuals with ever increased performance. Individual fitness is evaluated through competition with other individuals in the population. In other words, fitness signifies only the relative strengths of solutions; an increased fitness for one solution leads to a decreased fitness for another. This inverse fitness interaction will increase the capabilities of each population until the global optimal solution is attained [252]. Competitive coevolutionary models are especially suitable for problem domains where it is difficult to explicitly formulate an objective fitness function. The classic example of competitive coevolution is [133], which coevolved a population of sorting networks. Competitive coevolution has been since successfully applied to game playing strategies [231,210], evolving better pattern recognizers [153], coevolve complex agent behaviors [248], etc.

Cooperative Coevolution is inspired by the ecological relationship of symbiosis where different species live together in a mutually beneficial relationship. A general framework for cooperative coevolutionary algorithms has been introduced by Potter and De Jong [213] in 1994 for evolving solutions in the form of co-adapted subcomponents. Potter's model is usually applied in situations where a complex problem can be decomposed into a collection of easier sub-problems.[5] Each subproblem is assigned to a population, such that individuals in a given population represent potential components of a larger solution. Evolution of these populations occurs almost simultaneously, but in isolation to one another, interacting only to obtain fitness. Such a process can be static, in the sense that the divisions for the separate components are decided *a priori* and never altered, or dynamic, in the sense that populations of components may be added or removed as the run progresses [279]. This model has been analyzed from the evolutionary dynamics perspective in [171,279]. Cooperative CoEAs have had success in adversarial domains, e.g., designing artificial neural networks [212], multiobjective optimization [260], interaction frequency [211], etc. Some variants of Cooperative CoEAs have been proposed, such as co-evolutionary particle swarms [131] and coevolutionary differential evolution [246]. A combination of competitive and cooperative mechanisms has been proposed by Goh et al. [109] to solve multiobjective optimization problems in a dynamic environment.

Further, both styles of coevolution (i.e., competitive and cooperative) can use multiple, reproductively isolated populations; both can use similar patterns of inter-population interaction, similar diversity maintenance schemes, and so on. Aside from the novel problem-decomposition scheme of cooperative coevolution, the most salient difference between cooperative and competitive coevolution resides primarily in the game-theoretic properties of the domains to which these algorithms are applied [89].

3.2.4. Cultural algorithms

Cultural Algorithms (CA) are a class of computational models derived from observing the cultural evolution process in nature [225]. The term *culture* was first introduced by the anthropologist Edward B. Taylor in his book, *Primitive Culture*

[5] Problem decomposition consists in determining an appropriate number of subcomponents and the role each will play. The mechanism of dividing the optimization problem f into n sub-problems and treating them almost independently of one another strongly depends on properties of the function f.

[266]. Taylor offered a broad definition, stating that culture is "*that complex whole which includes knowledge, belief, art, morals, law, custom, and any other capabilities and habits acquired by man as a member of society*".

The term *cultural evolution* has been more recently used to refer to the idea that the processes producing cultural stability and change are analogous in important respects to those of biological evolution. In this view, just as biological evolution is characterized by changing frequencies of genes in populations through time as a result of such processes as natural selection, so cultural evolution refers to the changing distributions of cultural attributes in populations, likewise affected by processes such as natural selection but also by others that have no analog in genetic evolution. Using this idea, Reynolds developed a computational model in which cultural evolution is seen as an inheritance process that operates at both a *micro-evolutionary* level in terms of transmission of genetic material between individuals in a population and a *macro-evolutionary* level in terms of the knowledge acquired based upon individual experiences. Fundamental of the macro-evolutionary level is Renfrew's notion of individual's mental *mappa*, a cognitive map or worldview, that is based on experience with the external world and shapes interactions with it [218]. Individual mappa can be merged and modified to form *group mappa* in order to direct the future actions of the group and its individuals.

CAs consist of three components: (1) A *Population Space*, at the micro-evolutionary level, that maintains a set of individuals to be evolved and the mechanisms for its evaluation, reproduction, and modification. In population space, any of the evolutionary algorithms can be adopted and evolutionary operators aiming at a set of possible solutions to the problem are realized. (2) A *Belief Space*, at the macroevolutionary level, that represents the knowledge that has been acquired by the population during the evolutionary process. The main principle is to preserve beliefs that are socially accepted and discard unacceptable beliefs. There are at least five basic categories of cultural knowledge that are important in the belief space of any cultural evolution model: situational, normative, topographic or spatial, historical or temporal, and domain knowledge [227]. (3) The *Communications Protocol* is used to determine the interaction between the population and the beliefs.

The basic framework of a CA is shown in Fig. 12. In each generation, individuals in the population space are first evaluated using an evaluation or performance function (*Evaluate ()*). An *Acceptance* function (*Accept ()*) is then used to determine which of the individuals in the current population will be able to contribute with their knowledge to the belief space. Experiences of those selected individuals are then added to the contents of the belief space via function *Update ()*. The function *Generate ()* includes the influence of the knowledge from the belief space, through the *Influence ()* function, in the generation of offspring. The *Influence* function acts in such a way that the individuals resulting from the application of the variation operators (i.e., recombination and mutation) tend to approach the desirable behavior while staying away from undesirable behaviors. Such desirable and undesirable behaviors are defined in terms of the information stored in the belief space. The two functions *Accept ()* and *Influence ()* constitute the communication link between the population space and the belief space. This supports the idea of dual inheritance in that the population and the belief space are updated each time step based upon feedback from each other. Finally, in the replacement phase, a selection function (*Select ()*) is carried out from the current and the new populations. The CA repeats this process for each generation until the pre-specified termination condition is met.

As such, cultural algorithms are based on hybrid evolutionary systems that integrate evolutionary search and symbolic reasoning [258]. They are particularly useful for problems whose solutions require extensive domain knowledge (e.g., constrained optimization problems [56]) and dynamic environments (e.g., dynamic optimization problems [234]). The CA performance has been studied using benchmark optimization problems [226] as well as applied successfully in a number of diverse application areas, such as modeling the evolution of agriculture [224], job shop scheduling problem [230], re-engineering of Large-scale Semantic Networks [232], combinatorial optimization problems [196], multiobjective optimization problems [228], agent-based modeling systems [229], etc. Recently, many optimization methods have been combined with CAs, such as evolutionary programming [56], particle swarm optimization [165], differential evolution algorithm [24], genetic algorithm [282], and local search [195]. Adaptations of CAs have also been proposed (see for example [117] for multi-population CAs).

CA

```
1  Set the generation counter g = 0
2  Initialize the population (POP(g))
3  Initialize Belief Space (Beliefs(g))
4  repeat
5      Evaluate Population: Evaluate (POP(g))
6      Update(Beliefs(g), Accept(POP(g)))
7      Generate(POP(g), Influence (Beliefs(g)))
8      g = g + 1
9      Select(POP(g) from POP(g-1))
10 until a termination condition is achieved
```

Fig. 12. Cultural algorithm.

3.2.5. Scatter search and path relinking

Scatter Search (SS) and its Path Relinking (PR) generalization were originally developed in the late 1990s by F. Glover [106]. The fundamental concepts and principles of the method were first proposed in the 1970s [103] and were based on formulations, dating back to the 1960s [102], for combining decision rules and problem constraints. SS & PR differ from other evolutionary algorithms by providing unifying principles for joining solutions based on generalized path constructions (in both Euclidean and neighborhood spaces) and by utilizing strategic designs where other approaches resort to randomization. They are also intimately related to the Tabu Search metaheuristic, and derive additional advantages provided by intensification and diversification mechanisms that exploit adaptive memory. Interesting connections between the SS & PR approaches and the particle swarm optimization methodology introduced by Kennedy and Eberhart [145] have been identified in [174].

More explicitly, Scatter Search and Path Relinking operate on a set of solutions, the *reference set* (RefSet), by combining these solutions to create new ones. Typically, the reference set is relatively small. The algorithm starts with generating the initial set of solution vectors satisfying the criteria of diversity. The subset of the *best* vectors are then selected to be reference solutions. The notion of *best* is not limited to a measure given exclusively by the evaluation of the objective function but covers the diversity of solutions. In particular, a solution may be added to the reference set if the diversity of the set improves even when the objective value of the solution is inferior to that of other competing solutions. A set of new solutions – called *trial solutions* – is generated by means of structured combinations of subsets of the current reference solutions. An improvement procedure is then applied in order to try to improve the set of trial solutions. According to the result of such procedure, the reference set and even the population of solutions are updated to incorporate both high-quality and diversified solutions. The process is iterated until the reference set does not change anymore.

The main steps of the Scatter Search and Path Relinking algorithms are presented in Fig. 13 and are explained in the following:

- *SeedGeneration ()* method creates one or more seed solutions, which are arbitrary trial solutions used to initiate the remainder of the algorithm.
- *DiversificationGenerator ()* method generates a collection of diverse trial solutions from an arbitrary trial solution (or seed solution).
- *Improvement ()* method is used to transform a trial solution into one or more enhanced trial solutions. The local search is usually used as an improvement mechanism.
- *ReferenceSetUpdate ()* method is responsible for building and maintaining a reference set consisting of a number of *best* solutions found in the course of the algorithm. Solutions gain membership to the reference set according to their quality or their diversity.
- *SubsetGeneration ()* method operates on the reference set to produce a subset of its solutions as a basis for creating combined solutions.
- *SolutionCombination ()* method transforms a given subset of solutions produced by the *SubsetGeneration ()* method into one or more combined solution vectors. In SS, new solutions are created from linear combinations of subsets of the current reference solutions in Euclidean space. The linear combinations are chosen to produce points both inside and outside the convex regions spanned by the reference solutions. In PR, the process of generating linear combinations of a set of reference solutions is generalized to neighborhood spaces, rather than Euclidean space. Linear combinations of points

SSPL

```
1  Initial Phase:
2  SeedGeneration()
3  repeat
4      DiversificationGenerator()
5      Improvement()
6      ReferenceSetUpdate()
7  until The reference set is of cardinality n
8  Scatter Search/Path Relinking Phase:
9  repeat
10     SubsetGeneration()
11     SolutionCombination()
12     Improvement()
13     ReferenceSetUpdate()
14 until a termination condition is achieved
```

Fig. 13. Scatter search and path relinking.

in the Euclidean space can be reinterpreted as paths between and beyond solutions in a neighborhood space. The path between two solutions will generally yield solutions that share common attributes with the input solutions. To generate the desired paths, it is only necessary to select moves that perform the following role: upon starting from an *initiating solution*, the moves must progressively introduce attributes contributed by a *guiding solution* (or reduce the distance between attributes of the initiating and guiding solutions). The roles of the initiating and guiding solutions are interchangeable; each solution can also be induced to move simultaneously towards the other as a way of generating combinations. Multiparent path generation possibilities emerge in PR by considering the combined attributes provided by a set of guiding solutions, where these attributes are weighted to determine which moves are given higher priority.

SS & PR have been successfully applied to a wide range of applications, Glover et al. [108] provide overviews and a variety of references on these methods and the book on Scatter Search [158] provides the basic principles and fundamental ideas that will allow the readers to create successful applications of scatter search. Recent works on SS & PR are surveyed in [219].

3.3. Swarm intelligence

Swarm Intelligence (SI) is an innovative distributed intelligent paradigm for solving optimization problems that takes inspiration from the collective behavior of a group of social insect colonies and of other animal societies. SI systems are typically made up of a population of simple agents (an entity capable of performing/executing certain operations) interacting locally with one another and with their environment. These entities with very limited individual capability can jointly (cooperatively) perform many complex tasks necessary for their survival. Although there is normally no centralized control structure dictating how individual agents should behave, local interactions between such agents often lead to the emergence of global and self-organized behavior.

Several optimization algorithms inspired by the metaphors of swarming behavior in nature are proposed. Ant colony optimization, Particle Swarm Optimization, Bacterial foraging optimization, Bee Colony Optimization, Artificial Immune Systems and Biogeography-Based Optimization are examples to this effect.

Fundamentals of Computational Swarm Intelligence Book [81] introduces the reader to the mathematical models of social insects collective behavior and shows how they can be used in solving optimization problems. Another book by Chan et al. [44] aims at presenting recent developments and applications concerning optimization with SI, making a focus on Ant and Particle Swarm Optimization. Das et al. [61] provide a detailed survey of the state of the art research centered around the applications of SI algorithms in bioinformatics. The book by Abraham et al. [2] deals with the application of SI in data mining.

3.3.1. Ant colony optimization

Ant Colony Optimization (ACO) was introduced by M. Dorigo and colleagues [72,69,73] as a nature-inspired metaheuristic for the solution of hard combinatorial optimization problems. ACO takes inspiration from the foraging behavior of real ants. When searching for food, these ants initially explore the area surrounding their nest by performing a randomized walk. Along their path between food source and nest, ants deposit a chemical pheromone trail on the ground in order to mark some favorable path that should guide other ants to the food source [71]. After some time, the shortest path between the nest and the food source presents a higher concentration of pheromone and, therefore, attracts more ants. Artificial ant colonies exploited this characteristic of real ant colonies to build solutions to an optimization problem and exchange information on their quality through a communication scheme that is reminiscent of the one adopted by real ants [70].

Let us denote a combinatorial optimization problem by $P = (S, \Omega, f)$, where S is the search space defined by a finite set of decision variables X_i ($i = 1, \ldots, n$), Ω is a set of constraints among the variables and $f(\cdot) : S \mapsto \mathbb{R}^+$ is an objective function to be minimized. A feasible solution $s \in S$ is an assignment to each variable of a value in its domain such that all the problem constraints in Ω are satisfied.

ACO encodes a given combinatorial optimization problem instance as a *construction graph* $\mathcal{G}_{\mathcal{C}}(V, E)$, a fully connected graph whose nodes V are components of solutions, and edges E are connections between components. A solution to the given combinatorial optimization problem is encoded as a feasible walk on the construction graph $\mathcal{G}_{\mathcal{C}}$. We denote a solution component by c_i^j as the instantiation of a variable X_i with a particular value v_i^j. The definition of a solution component depends on the problem under consideration. In case of the popular example of Traveling Salesman Problem (TSP) [161], a component of the solution is a city that is added to a tour. Ants then need to appropriately combine solution components to form feasible walks. A pheromone value τ_{ij} is associated with each solution component c_i^j, this value serves as a form of memory, adapted over time to indicate the desirability of choosing solution component c_i^j. We denote the set of all solution components by $\mathcal{C}$ and the set of all pheromone trail parameters by $\mathcal{T}$.

The framework of a basic ACO metaheuristic is shown in Fig. 14. After initializing parameters and pheromone trails, the ACO algorithm iterates over three phases: ConstructAntSolutions, DeamonActions (optional) and UpdatePheromones.

Initialization: At the beginning of the algorithm, parameters are set and all pheromone variables are initialized to a value τ_0, which is a parameter of the algorithm.

ConstructAntSolutions: A set of m artificial ants incrementally and stochastically builds solutions to the considered problem starting from an initially empty partial solution $s_p = \emptyset$. At each construction step, the current partial solution s_p is extended by adding a feasible solution component c_i^j from the set $\mathcal{N}(s_p) \subseteq \mathcal{C}$. $\mathcal{C}$ denotes the set of all possible solution components and $\mathcal{N}(sp)$ is defined as the set of components that can be added to the current partial solution s_p while maintaining

ACO

1 Initialize pheromone values
2 **while** termination condition not met **do**
3 Construct Ants Solutions
4 Update Pheromones
5 Daemon Actions
6 **end**

Fig. 14. Ant colony optimization.

feasibility. In order to choose, which of the available solution components c_i^j should be added to the current partial solution s_p, a probabilistic choice is made. This decision is usually influenced by the amount of pheromone τ_{ij} associated with each of the elements of $\mathcal{N}(s_p)$, and by heuristic information about the problem. The most widely used rule for the stochastic choice is that originally proposed for Ant System (AS) [73] and given in Eq. (8).

$$p\left(c_i^j|s_p\right) = \frac{\tau_{ij}^{\alpha}\cdot\left[\eta\left(c_i^j\right)\right]^{\beta}}{\sum_{c_i^l\in\mathcal{N}(s_p)}\tau_{ij}^{\alpha}\cdot\left[\eta\left(c_i^l\right)\right]^{\beta}}, \forall c_i^j \in \mathcal{N}(s_p) \tag{8}$$

The heuristic information, denoted by the function $\eta(\cdot)$, assigns to each feasible solution component $c_i^j \in \mathcal{N}(s_p)$ a heuristic value. This value is used by the ants to make probabilistic decisions on how to move on the construction graph. The parameters α and β determine the relative respective influence of the pheromone values and the heuristic values on the decisions of the ant [75].

DaemonActions: Daemon actions refer to any centralized operation which cannot be performed by a single ant. The most used daemon action consists in the application of local search to the constructed solutions.

UpdatePheromones: The pheromone update process is intended to make solution components belonging to good solutions more desirable for ants operating in the following iterations. It consists of two mechanisms: *pheromone evaporation* and *pheromone deposit*. The *pheromone evaporation*, applied whilst constructing solutions, is the process by means of which the pheromone trail intensity on the components decreases over time. The goal is to make the solution components added to a partial walk on the construction graph less and less attractive as they are visited by ants. From a practical point of view, *pheromone evaporation* is needed to avoid a premature convergence of the algorithm to suboptimal solutions and then favoring the exploration of not yet visited areas of the search space. The *pheromone deposit* is applied after all ants have finished constructing a solution. The pheromone values are increased on solution components that are associated with a chosen set S_{upd} of high quality solutions. The goal is to make these solution components more attractive for ants in the following iterations. Many different schemes for pheromone update have been proposed within the ACO framework. The pheromone update is commonly implemented as [75]:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \sum_{s\in S_{upd}|c_i^j\in s} g(s) \tag{9}$$

where S_{upd} is the set of good solutions that are used to deposit pheromone, $g(\cdot): S \mapsto \mathbb{R}^+$ is a function such that $f(s) < f(\acute{s}) \Rightarrow g(s) \geqslant g(\acute{s})$ is commonly called the *quality function*, $0 < \rho \leqslant 1$ is the pheromone evaporation rate.

Different ACO algorithms have been proposed, all share the same characteristic idea. A survey on theoretical results on ACO and its most notable applications are discussed in [74,31,71,70,8]. The authors discussed the relations between ACO and other approximate methods for optimization, focused on some research efforts and identified some open questions. A recent overview of ACO [75] reveals that the majority of the currently published articles on ACO are clearly on its application to computationally challenging problems. The authors believe that ACO algorithms will show their greatest advantage when they will be systematically applied to real-world applications with time-varying data, multiple objectives, or when the availability of stochastic information about events or data is rather common.

3.3.2. Particle swarm optimization

Particle Swarm Optimization (PSO) was initially introduced in 1995 by James Kennedy and Russell Eberhart as a global optimization technique [145]. It uses the metaphor of the flocking behavior of birds to solve optimization problems. There are a number of differences between PSO and evolutionary optimization illustrated in [7], where some of the philosophical and performance differences are explored.

In PSO algorithm many autonomous entities (particles) are stochastically generated in the search space. Each particle is a candidate solution to the problem, and is represented by a velocity, a location in the search space and has a memory which helps it in remembering its previous best position. A swarm consists of N particles flying around in a D-dimensional search space. Moreover, every particle swarm has some sort of topology describing the interconnections among the particles. The

set of particles to which a particle i is topologically connected is called i's neighborhood. The neighborhood may be the entire population or some subset of it. Various topologies have been used to identify "some other particle" to influence the individual. The two most commonly used ones are known as *gbest* (for "global best") and *lbest* (for "local best"). The traditional particle swarm topology known as *gbest* was one where the best neighbor in the entire population influenced the target particle. While this may be conceptualized as a fully connected graph. The *lbest* topology, introduced in [79], is a simple ring lattice where each individual is connected to $K = 2$ adjacent members in the population array, with toroidal wrapping (naturally, this can be generalized to $K > 2$). Kennedy et al. [147] pointed out that the *gbest* topology had a tendency to converge very quickly with a higher chance of getting stuck in local optima. On the other hand, the *lbest* topology was slower but explored more fully, and typically ended up at a better optimum. The effects of various population topologies on the particle swarm algorithm were investigated in [148].

In the initialization phase of PSO, the positions and velocities of all individuals are randomly initialized. At each iteration, a particle i adjusts its position $\overrightarrow{X_i}$ and velocity $\overrightarrow{V_i}$ along each dimension d of the search space, based on the best position $\overrightarrow{P_i}$ it has encountered so far in its flight (also called the *personal best* for the particle) and the best position $\overrightarrow{P_g}$ found by any other particle in its topological neighborhood.

The velocity defines the direction and the distance the particle should go. It is updated according to the following equation:

$$V_{id}(t+1) = V_{id}(t) + C_1\varphi_1(P_{id}(t) - X_{id}(t)) + C_2\varphi_2(P_{gd}(t) - X_{id}(t)) \tag{10}$$

where $i = 1,2,\ldots,N$, and N is the size of the swarm; φ_1 and φ_2 are two random numbers uniformly distributed in the range $[0,1]$, C_1 and C_2 are constant multiplier terms known as *acceleration coefficients*. They represent the attraction that a particle has either towards its own success (the cognitive part) or towards the success of its neighbors (the social part), respectively.

The position of each particle is also updated in each iteration by adding the velocity vector to the position vector, i.e.,

$$X_{id}(t+1) = X_{id}(t) + V_{id}(t+1) \tag{11}$$

The general structure of the PSO algorithm can be summarized in Fig. 15.

In order to keep the particles from flying out of the problem space, Eberhart et al. [78] defined a clamping scheme to limit the velocity of each particle, so that each component of $\overrightarrow{V_i}$ is kept within the range $[-V_{max}, +V_{max}]$. The choice of the parameter V_{max} required some care since it appeared to influence the balance between exploration and exploitation. As has been noted in [7], the V_{max} particle swarm succeeds at finding optimal regions of the search space, but has no feature that enables it to converge on optima.

To overcome the problem of premature convergence of PSO, many strategies have been developed but by far the most popular are inertia and constriction. The inertia weight ω, introduced in [244], plays the role of balancing the global search and local search. It can be a positive constant or even a positive linear or nonlinear function of time. A large inertia weight encourages global exploration (i.e., diversifies the search in the whole search space) while a smaller inertia weight encourages local exploitation (i.e., intensifies the search in the current region) [245]. Using the inertia weight, the rule in Eq. (10) becomes:

$$V_{id}(t+1) = \omega V_{id}(t) + C_1\varphi_1(P_{id}(t) - X_{id}(t)) + C_2\varphi_2(P_{gd}(t) - X_{id}(t)) \tag{12}$$

PSO

```
Initialize a population of particles with random positions and velocities on D dimensions in the search space
while Terminating condition is not reached do
    for each particle i do
        Adapt velocity of the particle using Equation (10)
        Update the position of the particle using Equation (11)
        Evaluate the fitness f(X_i)
        if (f(X_i) < f(P_i)) then
            P_i ← X_i
        end
        if (f(X_i) < f(P_g)) then
            P_g ← X_i
        end
    end
end
```

Fig. 15. Particle swarm optimization.

Rather than applying inertia to the velocity memory, Clerc and Kennedy applied a constriction factor χ [52]. The velocity update scheme proposed by Clerc can be expressed for the dth dimension of ith particle as:

$$V_{id}(t+1) = \chi(V_{id}(t) + C_1\varphi_1(P_{id}(t) - X_{id}(t)) + C_2\varphi_2(P_{gd}(t) - X_{id}(t))) \tag{13}$$

The value of the constriction factor is calculated as follows:

$$\chi = \frac{2}{\varphi - 2 + \sqrt{\varphi^2 - 4\varphi}} \tag{14}$$

where $\varphi = \varphi_1 + \varphi_2 > 4$.

One of the drawbacks of the standard PSO is premature convergence and trapping in local optima. A great effort has been deployed to provide PSO convergence results through the stability analysis of the trajectories [198,52,270]. These studies were aimed at understanding theoretically how PSO algorithm works and why under certain conditions it might fail to find a good solution. Considerable research has been also conducted into further refinement of the original formulation of PSO in both continuous and discrete problem spaces [146], and areas such as dynamic environments [29], parallel implementation [16] and MultiObjective Optimization [223]. Modified versions of PSO based on diversity, mutation, crossover and efficient initialization using different distributions and low-discrepancy sequences are discussed in [199]. A large number of hybrid variants have been proposed, such as [269,268,136,80]. In 2008, Poli categorized a large number of publications dealing with PSO applications stored in the IEEE Xplore database [207].

So many papers related with the applications of PSO have been presented in the literature and several survey papers regarding these studies can be found in [147,51,208,17,262,38].

3.3.3. Bacterial foraging optimization algorithm

Bacterial Foraging Optimization Algorithm (BFOA), introduced by Passino in 2002 [200], is a relatively new paradigm for solving optimization problems, inspired by the social foraging behavior of *Escherichia coli* (*E. coli*) bacteria present in the human intestines. For many organisms, the survival-critical activity of foraging involves aggregations of organisms in groups, trying to find and consume nutrients in a manner that maximizes energy obtained from nutrient sources per unit time spent foraging, while at the same time minimizing exposure to risks from predators [201]. Foraging in groups, or social foraging, is a key element for avoiding predators and increasing the chance of finding food. A particularly interesting group foraging behavior has been demonstrated for several motile species of bacteria, including *E. coli* bacteria.

During foraging, individual bacteria move by taking small steps while searching for nutrients. Locomotion is achieved via a set of relatively rigid flagella that help an *E. coli* bacterium to move in alternating periods of swims and tumbles (tumbles serve to randomly reorient the bacteria). This alternation between the two modes is called chemotactic steps. Bacteria may respond directly to local physical cues such as concentration of nutrients or distribution of some chemicals (which may be laid by other individuals). They typically interact with other bacteria and with their growth substrata, such as solid surfaces, to give rise to complex behavior patterns. To facilitate the migration of bacteria in viscous substrates, such as semisolid agar surfaces, *E. coli* cells arrange themselves in a traveling ring and move over the surface of the agar in a coordinate manner called swarming motility. This is in contrast to swimming motility, which represents individual cell motility in aqueous environment [34]. After the bacterium has collected a sufficient amount of nutrients, it can self-reproduce and divide into two. The bacteria population can also suffer a process of elimination, through the appearance of a noxious substance, or to disperse, through the action of another substance, generating the effects of elimination and dispersion.

Based on these biological concepts, the BFOA is formulated on the basis of the following steps: chemotaxis, swarming, reproduction and elimination-dispersal. The general procedure of BFO algorithm is outlined in Fig. 16.

Chemotaxis: Chemotaxis is the process in which bacteria direct their movements according to certain chemicals in their environment. This is important for bacteria to find food by climbing up nutrient hills and at the same time avoid noxious substances. The sensors they use are receptor proteins which are very sensitive and possess high gain. That is, a small change in the concentration of nutrients can cause a significant change in behavior [167].

Suppose that we want to find the minimum of $J(\theta)$, where $\theta \in \mathbb{R}^D$ is the position of a bacterium in D-dimensional space and the cost function $J(\theta)$ is an attractant- repellant profile (i.e., it represents where nutrients and noxious substances are located). Then $J(\theta) \leqslant 0$ represents a nutrient rich environment, $J(\theta) = 0$ represents neutral medium and $J(\theta) > 0$ represents noxious substances. Let $\theta^i(j,k,l)$ represent ith bacterium at jth chemotactic, kth reproductive and lth elimination-dispersal step.

The position of the bacterium at the $(j+1)$th chemotactic step is calculated in terms of the position in the previous chemotactic step and the step size $C(i)$ (termed as run length unit) applied in a random direction $\phi(i)$:

$$\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i)\phi(i) \tag{15}$$

$\phi(i)$ is a unit length random direction to describe tumble and is given by:

$$\phi(i) = \frac{\Delta(i)}{\sqrt{\Delta^T(i)\Delta(i)}} \tag{16}$$

BFOA

Initialize parameters : $D, S, N_c, N_s, N_{re}, N_{ed}, P_{ed}, C(i), \theta^i\ (i = 1, 2, \ldots, S)$
while terminating condition is not reached **do**
  //Elimination-dispersal loop :
  for $l = 1, \ldots, N_{ed}$ **do**
    // Reproduction loop :
    for $k = 1, \ldots, N_{re}$ **do**
      //Chemotaxis loop :
      for $j = 1, \ldots, N_c$ **do**
        for each bacterium $i = 1, \ldots, S$ **do**
          Compute fitness function $J(i, j, k, l)$ using Equation (17)
          $J_{last} = J(i, j, k, l)$
          Tumble : Generate a random vector $\Delta(i) \in \mathbb{R}^D$
          Move : Compute the position of the bacterium $\theta^i(j + 1, k, l)$ at $j+1$th chemotactic step using Equation (15)
          Compute fitness function $J(i, j + 1, k, l)$ using Equation (17)
          Swim : $m = 0$ //counter for swim length
          while $m < N_s$ **do**
            $m = m + 1$
            if $J(i, j + 1, k, l) < J_{last}$ **then**
              $J_{last} = J(i, j + 1, k, l)$
              Move : Compute the position of the bacterium $\theta^i(j + 1, k, l)$ at $j+1$th chemotactic step using Equation (15)
              Compute fitness function $J(i, j + 1, k, l)$ using Equation (17)
            else
              $m = N_s$
            end
          end
        end
      end
      Reproduction :
      for $i = 1, \ldots, S$ **do**
        $J_{health}(i) = \sum_{j=1}^{Nc+1} J(i, j, k, l)$
      end
      Sort bacteria in order of ascending J_{health} (higher cost means lower health). The least healthy bacteria die and the other healthier bacteria split each into two bacteria, which are placed in the same location
    end
    Elimination-dispersal :
    for $i = 1, \ldots, S$ **do**
      Eliminate and disperse the ith bacterium, with probability P_{ed}
    end
  end
end

Fig. 16. Bacterial foraging optimization algorithm.

where $\Delta(i) \in \mathbb{R}^D$ is a randomly generated vector with elements within the interval $[-1, 1]$. The cost of each position is determined by the following equation:

$$J(i,j,k,l) = J(i,j,k,l) + J_{cc}(\theta, \theta^i(j,k,l)) \tag{17}$$

It can be noticed through Eq. (17) that the cost of a determined position $J(i,j,k,l)$ is also affected by the attractive and repulsive forces existing among the bacteria of the population given by J_{cc} (see Eq. (18)). If the cost at the location of the ith bacterium at $j + 1$th chemotactic step, denoted by $J(i,j + 1,k,l)$, is better (lower) than at the position $\theta^i(j,k,l)$ at the jth step, then

the bacterium will take another chemotactic step of size $C(i)$ in this same direction, up to a maximum number of permissible steps, called N_s.

Swarming: Swarming is a particular type of motility that is promoted by flagella and allows bacteria to move rapidly over and between surfaces and through viscous environments. Under certain conditions, cells of chemotactic strains of *E. coli* excrete an attractant, aggregate in response to gradients of that attractant, and form patterns of varying cell density. Central to this self-organization into swarm rings is chemotaxis. The cell-to-cell signaling in *E. coli* swarm may be represented by the following function:

$$\begin{aligned} Jcc(\theta, \theta^i(j,k,l)) = & \sum_{i=1}^{s}\left[-d_{attractant}\exp\left(-w_{attractant}\sum_{m=1}^{D}\left(\theta_m-\theta_m^i\right)^2\right)\right] \\ & +\sum_{i=1}^{s}\left[h_{repellant}\exp\left(-w_{repellant}\sum_{m=1}^{D}\left(\theta_m-\theta_m^i\right)^2\right)\right] \end{aligned} \tag{18}$$

where $\theta = [\theta_1, \theta_2, \ldots, \theta_D]^T$ is a point in the D-dimensional search space, $J_{cc}(\theta, \theta^i(j,k,l))$ is the objective function value that is to be added to the actual objective function, and $d_{attractant}$, $w_{attractant}$, $h_{repellant}$, $w_{repellant}$ are the coefficients which determine the depth and width of the attractant and the height and width of the repellant. These four parameters are to be chosen judiciously for a given problem. θ_m^i is the mth dimension of the position of the ith bacterium θ^i in the population of the S bacteria.

Reproduction: After N_c Chemotaxis steps (steps comprehending the movement and the cost determination of each bacterium position), the bacteria enter into the reproductive step. Suppose there are N_{re} reproduction steps. For reproduction, the least healthy bacteria will die, these are the bacteria that could not gather enough nutrients during the chemotactic steps, and will be replaced by the same number of healthy ones, thus the population size remains constant. The healthiest bacteria (those having sufficient nutrients and yielding lower values of fitness function) asexually split into two bacteria and will be placed in the same location.

Elimination and Dispersal: Changes in the environment can influence the proliferation and distribution of bacteria. So when local environmental change occurs, gradually (e.g. via consumption of nutrients) or suddenly due to some other reason (e.g. a significant local rise of temperature), all the bacteria in a region may die or disperse into some new part of the environment. This dispersal has the effect of destroying all the previous chemotactic processes. However, it may have good impact too, since dispersal may place bacteria into a nutrient rich region. Let N_{ed} be the number of elimination-dispersal events and, for each elimination-dispersal event, each bacterium in the population is subjected to elimination-dispersal with probability P_{ed}, in such a way that, at the end, the number of bacteria in the population remains constant (if a bacterium is eliminated, another one is dispersed to a random location).

In [62], the authors discussed some variations on the original BFOA algorithm and hybridizations of BFOA with other optimization techniques. They also provided an account of most of the significant applications of BFOA. However, experimentation with complex optimization problems reveal that the original BFOA algorithm possesses a poor convergence behavior compared to other nature-inspired algorithms, like GA and PSO, and its performance also heavily decreases with the growth of the search space dimensionality.

3.3.4. Bee colony optimization-based algorithms

Bee colony optimization-based algorithms are a new type of algorithm inspired by the behavior of honeybee colony that exhibits many features that can be used as models for intelligent systems and collective behavior. These features include *waggle dance* (communication), *food foraging*, *queen bee*, *task selection*, *collective decision making*, *nest site selection*, *mating* during flight and marriage in the bee colony, *floral/pheromone laying* and *navigation systems* [143]. Each model defines a given behavior for a specific task.

The structure of a bee hive. Honeybee colonies contain a single *queen* mated to a large number of males (*drones*) and thousands of *workers*. The queen is the only egg-laying female in a hive of bees, it secretes a pheromone that keeps all other females in the colony sterile. A fertile queen is able to selectively lay fertilized or unfertilized eggs. Fertilized eggs hatch into workers or virgin queens, while unfertilized eggs produce drones. The individual worker bees are always females because male drones do not contribute to social life apart from mating with queens during *nuptial flights*. The workers perform different tasks as nurses tending, nest-building, hive defense, and as foragers by collecting nectar and pollen to make honey and feed the brood.

Based on the queen bee concept, a handle of algorithms were developed in the literature, like Queen-bee Evolution Algorithm (QBE) [141] and Queen bee based crossover operator for GA [144].

Bee Dance and Communication. Bees exchange information about the location of food sources by performing a series of movements, often referred to as the *waggle dance*. Each hive has a so-called *dance floor area* in which the bees that have discovered nectar sources dance, in that way trying to *promote* food locations and persuade their nestmates to follow them. If a bee decides to leave the hive to collect nectar, it follows one of the bee dancers to one of the nectar areas.

Some bee colony based algorithms have been inspired by the communicative procedures of honey bees like Bee hive algorithm [278] and Discrete Bee Dance Algorithm [113].

Mating and marriage process. The queen bee is responsible for producing all of the eggs for the honey bee colony. Unlike organisms that mate again and again for the production of each offspring, the queen honey bee mates but once for a lifetime

of egg laying. This occurs in a series of mating flights far from the nest. During these flights, the queen bee will mate in the air with between seven and twenty drone bees. Sperm from the different drones will be deposited and accumulated in the queens' spermatheca to form the genetic pool of potential broods to be produced by the queen. For every fertilized egg that is laid by a queen, sperm is retrieved randomly from the mixture in its spermatheca. By mating with a number of drones, the queen is assured of bringing genetic diversity to the colony's offspring. Some of those diverse genes may be just the ones needed for the colony to survive in a changing environment.

The main algorithm proposed based on the process of marriage in real honey-bee is the Marriage in Honey Bees Optimization algorithm (MBO) [1]. In MBO algorithm, the mating-flight can be visualized as a set of transitions in a state-space (the environment) where the queen moves between the different states in the space in some speed and mates with the drone encountered at each state probabilistically. At the beginning of the mating flight, the queen is initialized with some energy content and returns to its nest either when its energy is depleted or when its spermatheca is full. A drone mates with a queen probabilistically using the following equation:

$$prob(Q,D) = e^{-dif/speed} \tag{19}$$

where $prob(Q,D)$ is the probability of adding the sperm of drone D to the spermatheca of queen Q (that is, the probability of a successful mating); dif represents the absolute difference between the fitness of D and the fitness of Q; $speed$ represents the flight speed of the queen.

After each transition in the space, the queen's speed and energy are reduced using the following equations:

$$speed(t+1) = \alpha * speed(t) \tag{20}$$

$$energy(t+1) = energy(t) - step \tag{21}$$

where α is a factor $\in]0,1[$ and $step$ is the amount of energy reduction after each transition.

The generic MBO algorithm is presented in Fig. 17.

Bee Foraging. The foraging behavior of honey bees has been extensively studied and is a useful example of self-organization. Scout-Bees in nature leave the hive and explore the areas around the colony's hive. They search for sources of pollen, nectar and propolis. Finishing the search, scout bees go back to the hive and transfer the collected nectar to receiver bees, which then store it in cells. Upon their return from a foraging trip, bees communicate the distance, direction, and quality of a flower site they have explored to their nestmates by making waggle dances on a dance floor inside the hive. The probability that a dance is followed by a naive bee is correlated with the duration of the dance. Thus dances for a source of high quality usually attract more followers than dances for a source of low quality. Typically a colony will know each day about a dozen or more potential food sources, each with its own level of profitability, determined by such variables as the distance

```
MBO
Initialize workers with some heuristic
Randomly generate the queens
Apply local search to get a good queen
for A pre-defined maximum number of mating-flights do
    for each Queen in the queen list do
        Initialize energy, speed and position
        The queen moves between states and probabilistically chooses drones
        if A drone is selected then
            Add its sperm to the queen's spermatheca (i.e., a list of partial solutions)
        end
        Update the queen's internal energy and speed
    end
    Generate broods by crossover and mutation
    Use workers to improve the broods
    Update workers' fitness
    while The best brood is better than the worst queen do
        Replace the least-fittest queen with the best brood
        Remove the best brood from the brood list
    end
end
```

Fig. 17. Marriage in honey bees optimization algorithm.

from the hive and the abundance and quality of the food. To gather its food efficiently, a colony must deploy its foragers among the flower patches in accordance with their profitabilities [242].

Foraging-inspired optimization algorithms make use of the bees' decentralized foraging behavior. During foraging honey bees balance exploitation of known food sources with exploration for new – and potentially better – food sources in a dynamic environment. Several foraging based algorithms have been proposed, such as the Bee Colony Optimization Metaheuristic (BCO) [169], Artificial Bee Colony Algorithm (ABC) [142] and Virtual Bee Algorithm (VBA) [283].

In the ABC algorithm, the bees in a colony are divided into three groups: *employed bees* (*forager bees*), *onlooker bees* (*observer bees*) and *scouts* (*explorer bees*). The first half of the colony consists of the employed bees and the second half includes the onlookers. The number of employed bees is equal to the number of food sources around the hive. The employed bee whose food source has been exhausted by the bees becomes a scout for searching new food sources randomly. An employed bee carries the information about food source and shares this information with a certain probability by waggle dance. On the other hand onlookers observe the waggle dance and so are placed on the food sources by using a probability based selection process. As the nectar amount of a food source increases, the probability value with which the food source is preferred by onlookers increases, too.

The main steps of the ABC algorithm are given in Fig. 18.

Nest site selection. As a bee colony becomes overcrowded, a third of the hive stays behind and rears a new queen, while a swarm of thousands departs with the old queen to produce a daughter colony. Many scout bees working in parallel explore for potential nest sites, advertise their discoveries to one another, engage in open deliberation, choose a final site, and navigate together, as a swirling cloud of bees, to their new home [241]. In contrast to foraging, where bees can typically forage at different locations simultaneously, nest-site selection always involves the selection of a single new site.

Algorithms that take their inspiration from the bee's nest site selection behavior apply the principle of decision-making process that enables a colony to identify and converge towards one best solution. Several studies, both experimental and theoretical, have investigated nest-site selection in honeybees [68,143].

Navigation. Bees can learn the directions and distances of their travels between hive and food sources by a process called *path integration*. A familiar food source is specified in the bee's memory by a vector encoding distance and direction from the hive. The directional component is defined by the sun compass and the distance component may be estimated by means of energy investment during flight. Also, bees can retrieve flight directions from landmarks when the sun compass is not available. This vector that is the output of the path integration process is used for navigation on subsequent trips to the food, and it is also what the bee encodes in its waggle dance. It thus appears that spatial navigation in bees, as in other animals and humans, is not a unitary process, but involves multiple navigational systems [178].

Some bee colony based algorithms have been designed with inspiration from honey bee's navigational system, like in [163].

Task selection. Task allocation operates without any central or hierarchical control to direct individuals into particular tasks. The queen does not issue commands and workers do not direct the behavior of other workers. Two kinds of factors determine what task an individual worker performs, and when it performs it: (1) *internal factors*, based on some attribute of the individual and (2) *external factors*, based on some environmental stimulus. One internal factor associated with task is worker age. Younger bees work inside the nest whereas older bees perform defensive and foraging tasks that occur outside the nest. This leads to a strongly aged-based division of labor. Another important factor affecting the tendency to perform a task has a genetic origin. A honey bee colony is characterized by high genetic diversity among its workers, generated by high levels of multiple mating by its queen. Honeybees from different patrilines vary in the rate at which they proceed from one task to the next as they grow older. In addition, honey bee colonies change their organizational structure in response to variation in colony conditions and the abundance of resources. For example, a honeybee forager's decision whether to collect nectar or remain in the nest depends on how much nectar is already stored in the nest. When extra clean-up work requires more nest-maintenance workers to be recruited from the reserves inside the nest, the current foragers are more likely to remain inside the nest [112].

ABC

1 Initialize Population
2 **repeat**
3 Place the employed bees on their food sources and determine their nectar amounts
4 Calculate the probability value of the sources with which they are preferred by the onlooker bees
5 Place the onlooker bees on the food sources depending on their nectar amounts
6 Stop the exploitation process of the sources exhausted by the bees
7 Send the scouts to the search area for discovering new food sources, randomly
8 Memorize the best food source found so far
9 **until** requirements are met

Fig. 18. Artificial bee colony algorithm.

Inspired by the cooperative behavior of social honey bees, some algorithms have been developed, among them are Decentralized Honey Bee algorithm [192], Swan [118] and Honey Bee Teamwork Strategy [233].

A recent literature survey of the algorithms inspired by bees' behavior in the nature and their applications is given in [143].

3.3.5. Artificial immune systems

The immune system is a network of cells, tissues, and organs that work together to defend the body against attacks by *foreign* invaders. The immune system protects organisms from pathogens (harmful micro-organisms such as bacteria and viruses) without prior knowledge of their structure. It has powerful learning and memory capabilities and presents an evolutionary type of response to infectious foreign elements [83]. This property, along with the highly distributed, adaptive, and self-organizing nature offers rich metaphors for its artificial counterpart. Artificial Immune Systems (AIS) attempts to apply immune system principles to optimization and machine learning problems [264].

There are several reviews of AIS research [127,287,263,128], and a number of books including [64] and [39] covering the field. The most recent and comprehensive survey on AIS is possibly that from Dasgupta et al. [65].

A few computational algorithms were developed and applied to several different types of problems in order to demonstrate how principles gleaned from the immune system can be used in the design of engineering tools for solving complex tasks. Four major AIS algorithms have been constantly developed and gained popularity: (1) negative selection algorithms; (2) artificial immune networks; (3) clonal selection algorithms; and (4) the danger theory and dendritic cell algorithms. Detailed discussion of these algorithms can be found in [264] and [65].

Negative Selection Based Algorithms. The key to a healthy immune system is its remarkable ability to distinguish between the body's own cells, recognized as '*self*', and foreign cells, or '*nonself*'. Negative selection is the main mechanism in the thymus that eliminates self-reactive cells, i.e. T-cells whose receptors recognize and bind with self antigens presented in the thymus. Thus, only T-cells that do not bind to self-proteins are allowed to leave the thymus. These matured T-cells then circulate throughout the body performing immunological functions and protecting the body against foreign antigens.

The negative selection algorithm is based on the principles of the self–nonself discrimination in the immune system and was initially introduced by Forrest et al. in 1994 [96] to detect data manipulation caused by a virus in a computer system. The starting point of this algorithm is to produce a set of self strings, S, that define the normal state of the system. The task then is to generate a set of detectors, D, that only bind/recognize the complement of S. These detectors can then be applied to new data in order to classify them as being self or non-self. This negative selection algorithm can be summarized in Fig. 19.

A diverse family of negative selection algorithms has been developed and has been extensively used in anomaly detection. A survey on negative selection algorithms was published in [138]. Some other researchers proposed negative selection algorithms that can be found in [264,65].

Clonal Selection Based Algorithms. The clonal selection theory postulates that a vast repertoire of different B-cells, each encoding antibodies with a predetermined shape and specificity, is generated prior to any exposure to an antigen. Exposure to an antigen then results in the proliferation or clonal expansion of only those B-cells with antibody receptors capable of reacting with part of the antigen. However, any clone of the activated B-cells with antigen receptors specific to molecules of the organism's own body (self-reactive receptors) is eliminated. Here, the affinity maturation of the B-cells takes place. During proliferation, a hypermutation mechanism becomes activated which diversifies the repertoire of B-cells. Antigen ensures that only those B-cells with high-affinity receptors are selected to differentiate into plasma cells and memory cells. Memory B-cells are developed to make a more effective immune response to antigens that had been encountered.

Many algorithms have been inspired by the adaptive immune mechanisms of B-cells [65]. The general one, named CLONALG [41], is based on clonal selection and affinity maturation principles. One cell generation in this algorithm includes the

NSA

input : S_{seen} = set of seen known self elements

output: D = set of generated detectors

```
repeat
    Randomly generate potential detectors and place them in a set P
    Determine the affinity of each member of P with each member of the self set S_seen
    if At least one element in S recognizes a detector in P according to a recognition threshold then
        The detector is rejected
    else
        The detector is added to the set of available detectors D
    end
until stopping criteria have been met
```

Fig. 19. Generic negative selection algorithm.

initiation of candidate solutions, selection, clone, mutation, reselection, and population replacement, which are somehow similar to evolutionary algorithms. When applied to pattern matching, CLONALG produces a set of memory antibodies M that match the members in a set S of patterns considered to be antigens. Fig. 20 outlines the working of CLONALG.

Many other clonal selection based algorithms have been introduced in the literature and have been applied to a wide range of optimization and clustering problems [128]. A summary of the basic features of these algorithms, their application areas and hybridization was published in [267].

Artificial Immune Networks. The immune Network theory, as originally proposed by Jerne [137], states that the immune system is a network in which antibodies, B-cells and T-cells recognize not only things that are foreign to the body, but also each other, creating a structurally and functionally plastic network of cells that dynamically adapts to stimuli over time. It is thus the interactions between cells that give rise to the emergence of complex phenomena such as memory [83] and other functionalities such as tolerance and reactivity [126]. The paper by Farmer et al. [83] is considered the pioneer work which inspired a variety of immune-network algorithms. One algorithm that has received much attention is aiNet first developed by de Castro and Von Zuben for the task of data clustering [67] and specialized into a series of algorithms for optimization and data-mining in a variety of domains over the following years [40,53]. AiNet is a simple extension of CLONALG but exploits interactions between B-cells according to the immune network theory. The aiNet algorithm is illustrated in Fig. 21.

A review of different artificial immune network models is presented in the paper by Galeano et al. [98]. Some other existing Immune network models can be found in [65].

Danger Theory inspired algorithms. The Danger theory attempts to explain the nature and workings of an immune response in a way different to the widely held self/nonself viewpoint. It does not deny the existence of self–nonself discrimination but rather states that the human immune system can detect danger in addition to antigens in order to trigger appropriate immune responses. Danger theory inspired algorithms are still in their infancy. The first paper that proposed an application of the Danger Theory was published in 2002 by Aickelin and Cayzer [4]. In 2003, Aickelin et al. proposed the Danger Project [3], an interdisciplinary project which aims at understanding from an immunological perspective the mechanisms of intrusion detection in the human immune system and applying these findings to AIS with a view to improving applications in computer security (see for example [116,149]).

Dendritic Cell algorithms. Dendritic cells (DCs) are immune cells that form part of the mammalian immune system. Their main function is to process antigen material and present it on the surface to other cells of the immune system, thus functioning as antigen-presenting cells and regulators of the adaptive immune system through the production of immunoregulatory cytokines (immune messenger proteins). DCs are responsible for some of the initial pathogenic recognition process, sampling the environment and differentiating depending on the concentration of signals, or perceived misbehavior, in the host tissue cells. Maturation of the immature DCs is regulated in response to various safe and danger signals. DCs can combine these signals with bacterial signatures (or PAMPs for Pathogen Associated Molecular Patterns Signals) to generate different output concentrations of costimulatory molecules, semi-mature cytokines and mature cytokines.

The dendritic cell algorithm (DCA) is based on the abstraction of the functionality of the biological DCs. It was first conceptualized and developed by Greensmith et al. [115] (see Fig. 22), which introduced the notion of danger signals, safe signals and PAMPs which all contribute to the context of a data signal at any given time.

As stated in [114], most of the works that applied DCA were related to computer security, but there are also applications in wireless sensor networks, robotics and scheduling of processes.

Over the last few years, important investigations have focused on the proposal of theoretical frameworks for the design of AIS [39]; theoretical investigations into existing AIS can be found in [99,265]. Other newly developed models have been recently reported in the literature, for example, Humoral Immune Response, and Pattern Recognition Receptor Model. The interested reader is referred to [65] for a detailed discussion of them.

CSA

input : S = set of patterns to be recognized, n the number of worst elements to select for removal

output: M = set of memory detectors capable of classifying unseen patterns

```
1 Create an initial random set of antibodies A
2 forall the Patterns in S do
3     Determine the affinity with each antibody in A
4     Generate clones of a subset of the antibodies in A with the highest affinity. The number of clones for an antibody is
      proportional to its affinity
5     Mutate attributes of these clones inversely proportional to its affinity.
6     Add these clones to the set A, and place a copy of the highest affinity antibodies in A into the memory set M
7     Replace the n lowest affinity antibodies in A with new randomly generated antibodies
8 end
```

Fig. 20. Generic clonal selection algorithm.

AIN

inputs : S = set of patterns to be recognized, nt network affinity threshold, ct clonal pool threshold, h number of highest affinity clones, a number of new antibodies to introduce

output: N = set of memory detectors capable of classifying unseen patterns

```
Create an initial random set of network antibodies, N
repeat
    forall the Patterns in S do
        Determine the affinity with each antibody in N
        Generate clones of a subset of the antibodies in N with the highest affinity. The number of clones for an antibody is proportional to its affinity
        Mutate attributes of these clones inversely proportional to its affinity, and place the h number of highest affinity clones into a clonal memory set, C
        Eliminate all members of C whose affinity with the antigen is less than a pre-defined threshold (ct)
        Determine the affinity amongst all the antibodies in C and eliminate those antibodies whose affinity with each other is less than a pre-specified threshold (ct)
        Incorporate the remaining clones in C into N
    end
    Determine the affinity between each pair of antibodies in N and eliminate all antibodies whose affinity is less than a pre-specified threshold nt
    Introduce a number (a) of new randomly generated antibodies into N
until a stopping condition has been met
```

Fig. 21. Generic immune network algorithm.

DCA

input : S = set of data items to be labelled safe or dangerous

output: L = set of data items labelled safe or dangerous

```
Create an initial population of dendritic cells (DCs), D
Create a set to contain migrated DCs, M
forall the Data items in S do
    Create a set of DCs randomly sampled from D, P
    forall the DCs in P do
        Add data items to DCs' collected list
        Update danger, PAMP and safe signal concentrations
        Update concentrations of output cytokines
        Migrate dendritic cell from D to M and create a new DC in D if concentration of costimulatory molecules is above a threshold
    end
end
forall the DCs in M do
    Set DC to be semi-mature if output concentration of semi-mature cytokines is greater than mature cytokines, otherwise set as mature
end
forall the Data items in S do
    Calculate number of times data item is presented by a mature DC and a semi-mature DC
    Label data item as safe if presented by more semi-mature DCs than mature DCs, otherwise label it as dangerous
    Add data item to labelled set M
end
```

Fig. 22. Generic dendritic cell algorithm.

3.3.6. Biogeography-based optimization

Biogeography-Based Optimization algorithm (BBO), developed by Dan Simon in 2008 [247], was strongly influenced by the equilibrium theory of island biogeography [173]. The basic premise of this theory is that the rate of change in the number of species on an island depends critically on the balance between the immigration of new species onto the island and the emigration of established species.

BBO algorithm maintains a set of candidate solutions called *islands* (or *habitats*), and each island feature is called a *Suitability Index Variable* (SIV). A quantitative performance index, called *Habitat Suitability Index* (HSI), is used as a measure of how good a solution is, which is analogous to *fitness* in other population-based optimization algorithms. The greater the total number of species on the island, which corresponds to a high HSI, the better the solution it contains. The number of species present on the island is determined by a balance between the rate at which the new species arrive and the rate at which the old species become extinct on the island. In BBO, each individual has its own immigration rate (λ) and emigration rate (μ). These parameters are affected by the number of species (S) in an island and are used to probabilistically share information between islands. Islands with smaller populations are more vulnerable to extinction (i.e. the immigration rate is high). But as more species inhabit the island, the immigration rate reduces and the emigration rate increases. In BBO, good solutions (i.e. islands with many species) tend to share their features with poor solutions (i.e. islands with few species), and poor solutions accept a lot of new features from good solutions. Maximum immigration rate (I) occurs when island is empty and decreases as more species are added and maximum emigration rate (E) occurs when all possible species S_{max} are present on the island. The immigration and emigration rates when there are S species in the habitat are given by Eq. (22):

$$\begin{aligned}\lambda_S &= I\left(1-\frac{S}{S_{max}}\right)\\ \mu_S &= E\left(\frac{S}{S_{max}}\right).\end{aligned} \tag{22}$$

For the sake of simplicity, the original BBO has considered a linear migration model where the immigration rate λ_S and the emigration rate μ_S are linear functions of the number of species S in the habitat, but different mathematical models of biogeography, that included more complex variables, are presented in [173]. There are, indeed, other important factors which influence migration rates between habitats, including the distance to the nearest neighboring habitat, the size of the habitat, climate (temperature and precipitation), plant and animal diversity, and human activity. These factors make immigration and emigration curves complicated, contrary to those described in the original BBO paper [247]. To verify the influence of different migration models on BBO performance, Haiping Ma [172] explores the behavior of six different migration models, and investigates performance through 23 benchmark functions with a wide range of dimensions and diverse complexities. Experimental results clearly show that different migration models in BBO result in significant changes in performance, and BBO migration models which are closer to nature (that is, nonlinear) are significantly better than linear models for most of the benchmarks.

We now consider the probability P_S that the habitat contains exactly S species. The number of species will change from time t to time $(t+\Delta t)$ as follows:

$$P_S(t+\Delta t) = P_S(t)(1-\lambda_S\Delta t-\mu_S\Delta t) + P_{S-1}\lambda_{S-1}\Delta t + P_{S+1}\mu_{S+1}\Delta t \tag{23}$$

which states that the number of species on the island in one time step is based on the total number of current species on the island, the new immigrants and the number of species who leave during the time period. We assume here that Δt is small enough so that the probability of more than one immigration or emigration can be ignored. In order to have S species at time $(t+\Delta t)$, one of the following conditions must hold:

- There were S species at time t, and no immigration or emigration occurred between t and $(t+\Delta t)$;
- One species immigrated onto an island already occupied by $(S-1)$ species at time t.
- One species emigrated from an island occupied by $(S+1)$ species at time t.

The limit of (23) as $\Delta t \to 0$ is given by Eq. (24).

$$\dot{P}_S = \begin{cases} -(\lambda_S+\mu_S)P_S + \mu_{S+1}P_{S+1} & \text{if } S=0\\ -(\lambda_S+\mu_S)P_S + \lambda_{S-1}P_{S-1} + \mu_{S+1}P_{S+1} & \text{if } 1\leqslant S\leqslant S_{max}-1\\ -(\lambda_S+\mu_S)P_S + \lambda_{S-1}P_{S-1} & \text{if } S=S_{max}\end{cases} \tag{24}$$

The BBO algorithm is overall described in Fig. 23.

A habitat's HSI can change suddenly due to apparently random events (unusually large flotsam arriving from a neighboring habitat, disease, natural catastrophes, etc.). BBO models this phenomena as *SIV mutation*, and uses species count probabilities to determine mutation rates. The species count probability P_S indicates the likelihood that a given solution S was expected *a priori* to exist as a solution for the given problem. In this context it should be remarked that very high *HSI* solutions and very low *HSI* solutions are both equally improbable. Medium *HSI* solutions are relatively probable. If a given solution has a low probability, then it is likely to mutate to some other solution. Conversely, a solution with high probability is less likely to mutate. Mutation is used to enhance the diversity of the population, thereby preventing the search from stag-

```
BBO
Initialize a set of solutions (habitats) to a problem
while termination condition not met do
    Evaluate the fitness (HSI) for each solution
    Compute S, λ and μ for each solution
    Modify habitats (Migration) based on λ and μ (see Figure 24)
    Mutation based on probability
    Implement elitism to retain the best solutions in the population from one generation to the next.
end
```

Fig. 23. Biogeography based optimization algorithm.

```
Migration
for i = 1 to NP do
    Use λ_i to probabilistically decide whether to immigrate to X_i
    if rand(0, 1) < λ_i then
        for j = 1 to NP do
            Select the emigrating island X_j with probability ∝ μ_j
            if rand(0, 1) < μ_j then
                Replace a randomly selected decision variable (SIV) of X_i with its corresponding variable in X_j
            end
        end
    end
end
```

Fig. 24. Migration.

nating. If a habitat S is selected to execute the mutation operation, then a chosen variable (*SIV*) is randomly modified based on its associated probability P_S. The mutation rate $m(S)$ is inversely proportional to the solution probability:

$$m(S) = m_{max}\left(1 - \frac{P_S}{P_{max}}\right) \tag{25}$$

where m_{max} is a user-defined parameter, and $P_{max} = \max_S P_S, S = 1, \ldots, S_{max}$.

Migration, described in Fig. 24, is used to modify existing islands by mixing features within the population, where $rand(0,1)$ is a uniformly distributed random number in the interval $[0,1]$ and X_{ij} is the jth SIV of the solution X_i. The BBO migration strategy is similar to the global recombination approach of evolutionary strategies (ES) [10], in which many parents can contribute to a single offspring. The main difference is that recombination is used to create new solutions, while in BBO migration is used to change existing solutions.

BBO has demonstrated good performance on various unconstrained and constrained benchmark functions. It has also been applied to real-world optimization problems, including sensor selection, economic load dispatch problems, satellite image classification, power system optimization, etc. The web site http://embeddedlab.csuohio.edu/BBO/ is dedicated to BBO and related material.

4. Discussion and conclusions

This work surveyed several important metaheuristic methods as they are described in the literature. Some of them are single-solution based, and others are population-based, and although they are based on different philosophies. Nevertheless, a number of these metaheuristics are implemented in a more and more similar way. A unified presentation of these methods is proposed under the name of *adaptive memory programming* (AMP) [256]. An important principle behind AMP is that a memory containing a set of visited solutions is kept, a new solution is constructed using the data in the memory and improved by a local search procedure or a more sophisticated metaheuristic, the improved solution is then used to update the memory.

Despite the lack of theoretical foundation, the advantages of metaheuristics are widely reported in the literature. However, there are a few general issues which should be addressed in order to exploit the metaheuristics to their full potential. The assessment of metaheuristics is commonly based on experimental comparisons. In this case, the use of descriptive

statistics, such as the sample mean and the standard deviation, is not sufficient. To ensure fair and meaningful comparisons of metaheuristics, different *statistical tests* may be carried out to analyze and compare the metaheuristics [50]. Many commercial (e.g., SAS/STAT, XPSS) and free softwares (e.g., R, MacAnova) are available to conduct such an analysis. Furthermore, it is important to recognize that the number of algorithm parameters has a direct effect on the complexity of the algorithm and on the number of parameter interactions, which complicates analysis. The importance of tuning metaheuristics is widely acknowledged in scientific literature since the successful application of metaheuristics to concrete problems requires the finding of a good initial parameter setting, which is a tedious and time consuming task. Attempts have already been made to reduce parameter tuning tasks by way of *adaptive metaheuristics*. Recent research reveals that the tuning problem has much in common with the problems that are typically faced in machine learning [27]. In addition, there is a clear need to provide *software frameworks for metaheuristics* that promote software reuse and reduce developmental effort. Such frameworks enable experts and developers to evaluate and compare fairly different algorithms, transform ready-to-use algorithms, design new algorithms, and combine and parallelize algorithms [258].

Another research trend is on solving large-scale optimization problems. In fact, the performance of most available optimization algorithms deteriorates very quickly when the dimensionality increases. Thus, scalability for high-dimensional problems becomes an essential requirement for sophisticated optimization algorithm approaches, including *parallel implementations* of well-known metaheuristics, as well as the adaptation of existing techniques for parallel architectures.

One of the most interesting trends in the last years is on *hybrid optimization methods*. Indeed, more and more papers are published about the hybridization of metaheuristics with other techniques for optimization. Hybridization is not restricted to the combination of different metaheuristics but includes the use of hybrid algorithms that combine local search or exact algorithms and metaheuristics [32]. Moreover, the combination of concepts from different metaheuristics and different research areas can lead to interesting new approaches, such as [187] which combines fuzzy logic and several optimization techniques. Such hybridizations can be used to take advantage of strengths from each algorithm in order to improve algorithms' performance for more effective and efficient problem-solving.

Despite widespread success of metaheuristics, there will always be questions related to the usefulness of a particular metaheuristic for solving a wide range of problems. The *No Free Lunch theorems* [281] state that all the black-box optimization algorithms[6] have the same average performance over the entire set of optimization problems. However, this does not prevent some algorithms from being better than others on particular classes of problems. Many theoretical studies on the analysis of landscapes (i.e., the topological structure over which search is being executed) of different optimization problems have shown that not only different problems correspond to different structures but also different instances of the same problem correspond to different structures. Consequently, understanding such structure is a first step towards understanding behavior of different search components of a metaheuristic, which can ultimately lead to design better search algorithms that incorporate more problem-specific knowledge. However, the efficiency of metaheuristics depends on the neighborhood operators provided by the user and the best alternative for a problem domain can only be formalized by an expert. *Hyper-heuristics*, which form an emerging search technology, provide a new approach to overcome the problem of such dependencies in metaheuristics. Hyper-heuristics are assumed to be problem independent and can be easily utilized by non-experts as well. The term has been defined to broadly describe the process of using (meta-) heuristics to choose the most appropriate (meta-) heuristics to solve the problem at hand.

The research community, the number of sessions, workshops, and conferences dealing with metaheuristics is growing significantly. Major conferences in the area include the Genetic and Evolutionary Computation Conference (GECCO), Metaheuristic International Conference (MIC), International Conference on Metaheuristics and Nature Inspired Computing (META), the IEEE Congress on Evolutionary Computation (CEC), etc.

References

[1] H.A. Abbass, MBO: marriage in honey bees optimisation: a haplometrosis polygynous swarming approach, in: CEC'2001 Congress on Evolutionary Computation, 2001, pp. 207–214.
[2] A. Abraham, C. Grosan, V. Ramos, Swarm Intelligence in Data Mining, Studies in Computational Intelligence, Springer, 2006.
[3] U. Aickelin, P. Bentley, S. Cayzer, J. Kim, J. Mcleod, Danger theory: the link between AIS and IDS? Artificial immune systems, in: Artificial Immune Systems, LNCS, Springer, 2003, pp. 147–155.
[4] U. Aikelin, S. Cayzer, The danger theory and its application to artificial immune systems, in: Proceedings of the 1st International Conference on Artificial Immune Systems, 2002, pp. 141–148.
[5] E. Alba, Parallel Metaheuristics: A New Class of Algorithms, Wiley-Interscience, 2005.
[6] E. Alba, J.M. Troya, A survey of parallel distributed genetic algorithms, Complexity 4 (1999) 31–52.
[7] P. Angeline, Evolutionary optimization versus particle swarm optimization: philosophy and performance differences, in: V. Porto, N. Saravanan, D. Waagen, A. Eiben (Eds.), Evolutionary Programming VII, Lecture Notes in Computer Science, vol. 1447, Springer, Berlin, Heidelberg, 1998, pp. 601–610.
[8] D. Angus, C. Woodward, Multiple objective ant colony optimisation, Swarm Intelligence 3 (2009) 69–85.
[9] A. Auger, N. Hansen, A restart CMA evolution strategy with increasing population size, in: B. McKay, et al. (Eds.), The 2005 IEEE International Congress on Evolutionary Computation (CEC'05), vol. 2, 2005, pp. 1769–1776.
[10] T. Bäck, Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms, Oxford University Press, Oxford, UK, 1996.
[11] T. Bäck, F. Hoffmeister, H.P. Schwefel, A survey of evolution strategies, in: Proceedings of the Fourth International Conference on Genetic Algorithms, 1991, pp. 2–9.

[6] A black-box optimization algorithm exploits limited knowledge concerning the optimization problem or the particular instance which is being solved.

[12] T. Bäck, G. Rudolph, H.P. Schwefel, Evolutionary programming and evolution strategies: Similarities and differences, in: Proceedings of the Second Annual Conference on Evolutionary Programming, 1993, pp. 11–22.
[13] T. Bäck, H.P. Schwefel, An overview of evolutionary algorithms for parameter optimization, Evolutionary Computation 1 (1993) 1–23.
[14] S. Baluja, Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning, Technical Report, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.
[15] S. Baluja, S. Davies, Using optimal dependency-trees for combinatorial optimization: learning the structure of the search space, in: D.H. Fisher (Ed.), 14th International Conference on Machine Learning, Morgan Kaufmann, 1997, pp. 30–38.
[16] A. Banks, J. Vincent, C. Anyakoha, A review of particle swarm optimization. Part i: background and development, Natural Computing 6 (2007) 467–484. doi:10.1007/s11047-007-9049-5.
[17] A. Banks, J. Vincent, C. Anyakoha, A review of particle swarm optimization. part ii: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications, Natural Computing 7 (2008) 109–124.
[18] R. Battiti, M. Brunato, F. Mascia, Reactive Search and Intelligent Optimization, Springer, 2008.
[19] R. Battiti, G. Tecchiolli, The reactive tabu search, ORSA Journal on Computing 6 (1994) 126–140.
[20] E.B. Baum, Towards practical "neural" computation for combinatorial optimization problems, in: AIP Conference Proceedings: Neural Networks for Computing, Snowbird, UT, USA, 1986, pp. 53–58.
[21] J. Baxter, Local optima avoidance in depot location, Journal of the Operational Research Society 32 (1981) 815–819.
[22] D. Beasley, D. Bull, R.R. Martin, An overview of genetic algorithms. Part i: fundamentals, University Computing 15 (1993) 58–69.
[23] D. Beasley, D. Bull, R.R. Martin, An overview of genetic algorithms. Part ii: research topics, University Computing 15 (1993) 170–181.
[24] R.L. Becerra, C.A.C. Coello, A cultural algorithm with differential evolution to solve constrained optimization problems, in: IBERAMIA, 2004, pp. 881–890.
[25] H.G. Beyer, H.P. Schwefel, Evolution strategies – a comprehensive introduction, Natural Computing 1 (2002) 3–52.
[26] L. Bianchi, M. Dorigo, L.M. Gambardella, W.J. Gutjahr, A survey on metaheuristics for stochastic combinatorial optimization, Natural Computing 8 (2009) 239–287.
[27] M. Birattari, Tuning Metaheuristics: A Machine Learning Perspective, Springer Publishing Company, Incorporated, first ed., 2005 (2nd printing edition, 2009).
[28] M. Birattari, L. Paquete, T. Stützle, K. Varrentrapp, Classification of Metaheuristics and Design of Experiments for the Analysis of Components, Technical Report AIDA-01-05, FG Intellektik, FB Informatik, Technische Universität Darmstadt, Darmstadt, Germany, 2001.
[29] T. Blackwell, Particle swarm optimization in dynamic environments, in: S. Yang, Y.S. Ong, Y. Jin (Eds.), Evolutionary Computation in Dynamic and Uncertain Environments, Studies in Computational Intelligence, vol. 51, Springer, Berlin, Heidelberg, 2007, pp. 29–49.
[30] T. Blickle, L. Thiele, A comparison of selection schemes used in genetic algorithms, Evolutionary Computation 4 (1995) 311–347.
[31] C. Blum, Ant colony optimization: Introduction and recent trends, Physics of Life Reviews 2 (2005) 353–373.
[32] C. Blum, J. Puchinger, G.R. Raidl, A. Roli, Hybrid metaheuristics in combinatorial optimization: a survey, Applied Soft Computing 11 (2011) 4135–4151.
[33] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: overview and conceptual comparison, ACM Computing Surveys 35 (2003) 268–308.
[34] M.P. Brenner, L.S. Levitov, E.O. Budrene, Physical mechanisms for chemotactic pattern formation by bacteria, Biophysical Journal 74 (1998) 1677–1693.
[35] J. Brest, M. Maucec, Self-adaptive differential evolution algorithm using population size reduction and three strategies, soft computing – a fusion of foundations, Methodologies and Applications 15 (2011) 2157–2174.
[36] J. Brownlee, Clever Algorithms: Nature-Inspired Programming Recipes, Clever Algorithms: Nature-Inspired Programming Recipes, Lulu, 2011, pp. 29–86.
[37] E. Carrizosa, M. Drazic, Z. Drazic, N. Mladenovic, Gaussian variable neighborhood search for continuous optimization, Computers & Operations Research 39 (2012) 2206–2213.
[38] O. Castillo, P. Melin, Optimization of type-2 fuzzy systems based on bio-inspired methods: a concise review, Information Sciences 205 (2012) 1–19.
[39] L.N. de Castro, Artificial Immune Systems: A New Computational Intelligence Approach, Springer-Verlag, London, 2002.
[40] L.N. de Castro, F.J. Von Zuben, aiNet: an artificial immune network for data analysis, in: H.A. Abbass, R.A. Sarker, C.S. Newton (Eds.), Data Mining: A Heuristic Approach, Idea Group Publishing, 2001, pp. 231–259.
[41] L.N. de Castro, F.J. Von Zuben, Learning and optimization using the clonal selection principle, IEEE Transactions on Evolutionary Computation 6 (2002) 239–251.
[42] V. Cerny, Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm, Journal of Optimization Theory and Applications 45 (1985) 41–51.
[43] U. Chakraborty, Advances in Differential Evolution, first ed., Springer Publishing Company, 2008.
[44] F.T.S. Chan, M. Tiwari, Swarm Intelligence, Focus on Ant and Particle Swarm Optimization, I-Tech Education and Publishing, Vienna, Austria, 2007.
[45] I. Charon, O. Hudry, The noising method: a new method for combinatorial optimization, Operations Research Letters 14 (1993) 133–137.
[46] I. Charon, O. Hudry, The noising methods: a generalization of some metaheuristics, European Journal of Operational Research 135 (2001) 86–101.
[47] I. Charon, O. Hudry, The noising methods: a survey, in: P. Hansen, C.C. Ribeiro (Eds.), Essays and Surveys in Metaheuristics, Kluwer Academic Publishers, 2002, pp. 245–261.
[48] I. Charon, O. Hudry, Self-tuning of the noising methods, Optimization 58 (2009) 1–21.
[49] R. Chelouah, P. Siarry, Tabu search applied to global optimization, European Journal of Operational Research 123 (2000) 256–270.
[50] M. Chiarandini, L. Paquete, M. Preuss, E. Ridge, Experiments on Metaheuristics: Methodological Overview and Open Issues, Technical Report DMF-2007-03-003, The Danish Mathematical Society, Denmark, 2007.
[51] M. Clerc, Particle Swarm Optimization, ISTE, London, UK, 2006.
[52] M. Clerc, J. Kennedy, The particle swarm – explosion, stability, and convergence in a multidimensional complex space, IEEE Transactions on Evolutionary Computation 6 (2002) 58–73.
[53] G.P. Coelho, F.V. Zuben, omni-aiNet: an immune-inspired approach for omni optimization, in: Proceedings of the 5th International Conference on Artificial Immune Systems, Springer, 2006, pp. 294–308.
[54] C. Coello Coello, G.B. Lamont, D.A. Van Veldhuizen, Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation), Springer Verlag New York, Inc., Secaucus, NJ, USA, 2006.
[55] C.A. Coello Coello, Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art, Computer Methods in Applied Mechanics and Engineering 191 (2002) 1245–1287.
[56] C.A. Coello Coello, R.L. Becerra, Adding knowledge and efficient data structures to evolutionary programming: a cultural algorithm for constrained optimization, in: GECCO, 2002, pp. 201–209.
[57] N.E. Collins, R.W. Eglese, B. Golden, Simulated annealing – an annotated bibliography, American Journal of Mathematical and Management Sciences 8 (1988) 209–307.
[58] J. Courat, G. Raynaud, I. Mrad, P. Siarry, Electronic component model minimization based on log simulated annealing, IEEE Transactions on Circuits and Systems: Part I 41 (1994) 790–795.
[59] M. Creutz, Microcanonical Monte Carlo simulation, Physical Review Letters 50 (1983) 1411–1414.
[60] D. Cvijovic, J. Klinowski, Tabu search: an approach to the multiple minima problem, Science 267 (1995) 664–666.
[61] S. Das, A. Abraham, A. Konar, Swarm intelligence algorithms in bioinformatics, in: Computational Intelligence in Bioinformatics, Studies in Computational Intelligence, vol. 94, Springer, 2008, pp. 113–147.

[62] S. Das, A. Biswas, S. Dasgupta, A. Abraham, Bacterial foraging optimization algorithm: theoretical foundations, analysis, and applications, in: A. Abraham, A.E. Hassanien, P. Siarry, A. Engelbrecht (Eds.), Foundations of Computational Intelligence, Studies in Computational Intelligence, vol. 3, Springer, Berlin, Heidelberg, 2009, pp. 23–55.
[63] S. Das, P.N. Suganthan, Differential evolution: a survey of the state-of-the-art, IEEE Transactions on Evolutionary Computation 15 (2011) 4–31.
[64] D. Dasgupta, Artificial Immune Systems and Their Applications, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.
[65] D. Dasgupta, S. Yu, F. Nino, Recent advances in artificial immune systems: models and applications, Applied Soft Computing 11 (2011) 1574–1587.
[66] J. De Bonet, C. Isbell, P. Viola, Mimic: finding optima by estimating probability densities, in: NIPS, 1996, pp. 424–430.
[67] L.N. de Castro, F.J.V. Zuben, An evolutionary immune network for data clustering, in: Proceedings of the 6th Brazilian Symposium on Neural Networks, IEEE Computer Society Press, 2000, pp. 84–89.
[68] K. Diwold, M. Beekman, M. Middendorf, Honeybee optimisation – an overview and a new bee inspired optimisation scheme, in: A. Doe (Ed.), Handbook of Swarm Intelligence, Springer, Berlin, Heidelberg, 2010, pp. 295–327.
[69] M. Dorigo, Optimization, Learning and Natural Algorithms, Ph.D. Thesis, Politecnico di Milano, Italy, 1992.
[70] M. Dorigo, M. Birattari, T. Stützle, U. Libre, D. Bruxelles, A.F.D. Roosevelt, Ant colony optimization – artificial ants as a computational intelligence technique, IEEE Computational Intelligence Magazine 1 (2006) 28–39.
[71] M. Dorigo, C. Blum, Ant colony optimization theory: a survey, Theoretical Computer Science 344 (2005) 243–278.
[72] M. Dorigo, V. Maniezzo, A. Colorni, Positive Feedback as a Search Strategy, Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1991.
[73] M. Dorigo, V. Maniezzo, A. Colorni, The ant system: optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man, and Cybernetics – Part B 26 (1996) 29–41.
[74] M. Dorigo, T. Stützle, The ant colony optimization metaheuristic: algorithms, applications, and advances, in: F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics, International Series in Operations Research & Amp; Management Science, vol. 57, Springer, New York, 2003, pp. 250–285.
[75] M. Dorigo, S. Thomas, Ant colony optimization: overview and recent advances, in: M. Gendreau, J.Y. Potvin (Eds.), Handbook of Metaheuristics, International Series in Operations Research & Management Science, Springer, US, 2010, pp. 227–263.
[76] J. Dréo, A. Pétrowski, P. Siarry, E. Taillard, Metaheuristics for Hard Optimization: Methods and Case Studies, Springer, 2006.
[77] G. Dueck, T. Scheuer, Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing, Journal of Computational Physics 90 (1990) 161–175.
[78] R. Eberhart, P. Simpson, R. Dobbins, Computational Intelligence PC Tools, Academic Press Professional, Inc., San Diego, CA, USA, 1996.
[79] R.C. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, EP '98, Piscataway: IEEE, Nagoya, Japan, 1995, pp. 39–43.
[80] A. El Dor, M. Clerc, P. Siarry, Hybridization of differential evolution and particle swarm optimization in a new algorithm: DEPSO-2S, in: Proceedings of the 2012 International Conference on Swarm and Evolutionary Computation, ICAISC (SIDE-EC), Zakopane, Poland, 2012, pp. 57–65.
[81] A. Engelbrecht, Fundamentals of Computational Swarm Intelligence, Wiley, 2006.
[82] R. Etxeberria, P. Larrañaga, Global Optimization Using Bayesian Networks, in: A.A.O. Rodriguez, M.R.S. Ortiz, R.S. Hermida (Eds.), CIMAF 99, Second Symposium on Artificial Intelligence, Adaptive Systems, La Habana, 1999, pp. 332–339.
[83] J.D. Farmer, N.H. Packard, A.S. Perelson, The immune system, adaptation, and machine learning, Physica D 2 (1986) 187–204.
[84] T.A. Feo, M.G.C. Resende, A probabilistic heuristic for a computationally difficult set covering problem, Operations Research Letters 8 (1989) 67–71.
[85] T.A. Feo, M.G.C. Resende, Greedy randomized adaptive search procedures, Journal of Global Optimization 6 (1995) 109–133.
[86] P. Festa, M. Resende, An annotated bibliography of GRASP, Part I: algorithms, International Transactions in Operational Research 16 (2009) 1–24.
[87] P. Festa, M. Resende, An annotated bibliography of GRASP, Part II: applications, International Transactions in Operational Research 16 (2009) 131–172.
[88] P. Festa, M.G.C. Resende, Hybridizations of GRASP with Path-Relinking, Technical Report HYB-GPR-2011, AT&T Labs Research, 2011.
[89] S.G. Ficici, Solution Concepts in Coevolutionary Algorithms, Ph.D. Thesis, Brandeis University, Waltham, MA, USA, 2004. AAI3127125.
[90] M. Fleischer, Simulated annealing: past, present and future, in: Proceedings of the 27th Conference on Winter Simulation, Arlington, VA, USA, 1995, pp. 155–161.
[91] D.B. Fogel, System Identification through Simulated Evolution: A Machine Learning Approach to Modeling, Ginn Press, 1991.
[92] D.B. Fogel, Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, IEEE Press, Piscataway, NJ, USA, 1995.
[93] D.B. Fogel, L.J. Fogel, J. Atma, Meta-evolutionary programming, in: R. Chen (Ed.), Proceedings of 25th Asilomar Conference on Signals, Systems & Computers, Pacific Grove, CA, 1991, pp. 540–545.
[94] D.B. Fogel, L.J. Fogel, J. Atma, G. Fogel, Hierarchic methods of evolutionary programming, in: D. Fogel, W. Atmar (Eds.), Proceedings of the First Ann. Conf. on Evolutionary Programming, Evolutionary Programming Society, La Jolla, CA, 1992, pp. 175–182.
[95] L.J. Fogel, A.J. Owens, M.J. Walsh, Artificial Intelligence through Simulated Evolution, John Wiley, New York, USA, 1966.
[96] S. Forrest, A.S. Perelson, L. Allen, R. Cherukuri, Self-nonself discrimination in a computer, Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on, in: Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on, 1994, pp. 202–212.
[97] A.A. Freitas, A survey of evolutionary algorithms for data mining and knowledge discovery, in: Advances in Evolutionary Computing, Springer-Verlag New York, Inc., New York, NY, USA, 2003, pp. 819–845.
[98] J.C. Galeano, A. Veloza-Suan, F.A. González, A comparative analysis of artificial immune network models, in: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, GECCO '05, ACM, New York, NY, USA, 2005, pp. 361–368.
[99] S.M. Garrett, How do we evaluate artificial immune systems?, Evolutionary Computation 13 (2005) 145–177
[100] M. Gendreau, Chapter 2: an introduction to tabu search, in: F. Glover, G.A. Kochenberger (Eds.), Handbook of Metaheuristics, Kluwer Academic Publishers, 2003, pp. 37–54.
[101] M. Gendreau, J.Y. Potvin, Chapter 6: Tabu search, in: E.K. Burke, G. Kendall (Eds.), Search Methodologies, Springer, 2006, pp. 165–186.
[102] F. Glover, Parametric combinations of local job shop rules, in: ONR Research Memorandum, No. 117, GSIA, Carnegie Mellon University, Pittsburgh, PA, 1963.
[103] F. Glover, Heuristics for integer programming using surrogate constraints, Decision Sciences 8 (1977) 156–166.
[104] F. Glover, Future paths for integer programming and links to artificial intelligence, Computers and Operations Research 13 (1986) 533–549.
[105] F. Glover, Tabu search for nonlinear and parametric optimization (with links to genetic algorithms), Discrete Applied Mathematics 49 (1994) 231–255.
[106] F. Glover, A template for scatter search and path relinking, Lecture Notes on Computer Science 1363 (1997) 13–54.
[107] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, 1997.
[108] F. Glover, M. Laguna, R. Marti, Scatter search and path relinking: advances and applications, in: F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 57, Springer, New York, 2003, pp. 1–35.
[109] C.K. Goh, K.C. Tan, A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization, Transactions on Evolutionary Computation 13 (2009) 103–127.
[110] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine learning, first ed., Studies in Computational Intelligence, Addison-Wesley Longman Publishing Co., Inc., 1989.
[111] D.E. Goldberg, K. Deb, A comparative analysis of selection schemes used in genetic algorithms, in: Foundations of Genetic Algorithms, Morgan Kaufman, 1991, pp. 69–93.

[112] D.M. Gordon, The organization of work in social insect colonies, Complexity 8 (2002) 43–46.
[113] N. Gordon, I.A. Wagner, A.M. Brucks, Discrete bee dance algorithms for pattern formation on a grid, in: Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology, IAT '03, IEEE Computer Society, Washington, DC, USA, 2003, pp. 545–549.
[114] J. Greensmith, U. Aickelin, The deterministic dendritic cell algorithm, in: Artificial Immune Systems, LNCS, Springer, 2008, pp. 291–302.
[115] J. Greensmith, U. Aickelin, S. Cayzer, Introducing dendritic cells as a novel immune-inspired algorithm for anomaly detection, artificial immune systems, in: Artificial Immune Systems, LNCS, Springer, 2005, pp. 153–167.
[116] J. Greensmith, U. Aickelin, J. Twycross, Detecting danger: applying a novel immunological concept to intrusion detection systems, in: Proceedings 6th International Conference in Adaptive Computing in Design and Manufacture (ACDM2004), Bristol, UK, 2004.
[117] Y. Guo, J. Cheng, Y. Cao, Y. Lin, A novel multi-population cultural algorithm adopting knowledge migration, Soft Computing 15 (2011) 897–905.
[118] A. Gupta, N. Koul, Swan: a swarm intelligence based framework for network management of ip networks, Proceedings of the International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007), Vol. 01, IEEE Computer Society, Washington, DC, USA, 2007, pp. 114–118.
[119] N. Hansen, The CMA evolution strategy: a comparing review, in: J. Lozano, P. Larranaga, I. Inza, E. Bengoetxea (Eds.), Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms, Springer, 2006, pp. 75–102.
[120] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evolutionary Computation 9 (2001) 159–195.
[121] N. Hansen, A. Ostermeier, A. Gawelczyk, On the adaptation of arbitrary normal mutation distributions in evolution strategies: the generating set adaptation, in: Proceedings of the 6th International Conference on Genetic Algorithms, Morgan Kaufman Publishers Inc., San Francisco, CA, USA, 1995, pp. 57–64.
[122] P. Hansen, N. Mladenovic, J.A.M. Pérez, Variable neighbourhood search: methods and applications, 4OR 6 (2008) 319–360.
[123] P. Hansen, N. Mladenovic, J.A.M. Pérez, Variable neighbourhood search: algorithms and applications, Annals of Operations Research 175 (2010) 367–407.
[124] G. Harik, Linkage Learning via Probabilistic Modeling in the EcGA, IlliGAL Report 99010, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 1999.
[125] G. Harik, F. Lobo, D. Goldberg, The compact genetic algorithm, in: Proceedings of the IEEE Conference on Evolutionary Computation, 1998, pp. 523–528.
[126] E. Hart, H. Bersini, F. Santos, Structure versus function: a topological perspective on immune networks, Natural Computing 9 (2010) 603–624.
[127] E. Hart, C. McEwan, J. Timmis, A. Hone, Advances in artificial immune systems, Evolutionary Intelligence 4 (2011) 67–68.
[128] E. Hart, J. Timmis, Application areas of AIS: the past, the present and the future, Applied Soft Computing 8 (2008) 191–201.
[129] J.P. Hart, A.W. Shogan, Semi-greedy heuristics: an empirical study, Operations Research Letters 6 (1987) 107–114.
[130] M. Hauschild, M. Pelikan, An introduction and survey of estimation of distribution algorithms, Swarm and Evolutionary Computation 1 (2011) 111–128.
[131] Q. He, L. Wang, An effective co-evolutionary particle swarm optimization for constrained engineering design problems, Engineering Applications of Artificial Intelligence 20 (2007) 89–99.
[132] M. Herdy, Reproductive isolation as strategy parameter in hierarichally organized evolution strategies, in: PPSN, 1992, p. 209.
[133] W.D. Hillis, Co-evolving parasites improve simulated evolution as an optimization procedure, Physica D 42 (1990) 228–234.
[134] M.J. Hirsch, C.N. Meneses, P.M. Pardalos, M.G.C. Resende, Global optimization by continuous GRASP, Optimization Letters 1 (2007) 201–212.
[135] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, USA, 1975.
[136] L. Idoumghar, M. Idrissi-Aouad, M. Melkemi, R. Schott, Hybrid PSO-SA type algorithms for multi-modal function optimization and reducing energy consumption in embedded systems, Journal of Applied Computational Intelligence and Soft Computing 2011 (2011) 1–12.
[137] N.K. Jerne, Towards a network theory of the immune system, Annals of Immunology 125C (1973) 373–389.
[138] Z. Ji, D. Dasgupta, Revisiting negative selection algorithms, Evolutionary Computation 15 (2007) 223–251.
[139] D.S. Johnson, Local optimization and the travelling salesman problem, in: Proceedings of the 17th Colloquium on Automata, Languages, and Programming, Warwick, England, 1990, pp. 446–461.
[140] D.S. Johnson, L.A. McGeoch, The travelling salesman problem: a case study in local optimization, in: E.H.L. Aarts, J.K. Lenstra (Eds.), Local Search in Combinatorial Optimization, John Wiley & Sons, 1997, pp. 215–310.
[141] S. Jung, Queen-bee evolution for genetic algorithms, Electronics Letters 39 (2003) 575–576.
[142] D. Karaboga, An Idea Based on Honey Bee Swarm for Numerical Optimization, Technical Report TR06, Erciyes University, 2005.
[143] D. Karaboga, B. Akay, A survey: algorithms simulating bee swarm intelligence, Artificial Intelligence Review 31 (2009) 61–85.
[144] A. Karci, Imitation of bee reproduction as a crossover operator in genetic algorithms, in: PRICAI, 2004, pp. 1015–1016.
[145] J. Kennedy, R. Eberhart, Particle swarm optimization, IEEE International Conference on Neural Networks 4 (1995) 1942–1948.
[146] J. Kennedy, R. Eberhart, Discrete binary version of the particle swarm algorithm, in: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, vol. 5, 1997, p. 4104–4108.
[147] J. Kennedy, R. Eberhart, Y. Shi, Swarm Intelligence, Morgan Kaufman, San Francisco, 2001.
[148] J. Kennedy, R. Mendes, Population structure and particle swarm performance, Proceedings of the World on Congress on Computational Intelligence 2 (2002) 1671–1676.
[149] J. Kim, J. Greensmith, J. Twycross, U. Aickelin, Malicious Code Execution Detection and Response Immune System Inspired by the Danger Theory, CoRR abs/1003.4142, 2010.
[150] S. Kirkpatrick, C. Gelatt, M. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[151] A. Konak, D. Coit, A. Smith, Multi-objective optimization using genetic algorithms: a tutorial, Reliability Engineering & System Safety in Special Issue – Genetic Algorithms and Reliability 92 (2006) 992–1007.
[152] C. Koulamas, S.R. Antony, R. Jaen, A survey of simulated annealing applications to operations research problems, Omega 22 (1994) 41–56.
[153] T. Kowaliw, N.N. Kharma, C. Jensen, H. Moghnieh, J. Yao, Using competitive co-evolution to evolve better pattern recognisers, International Journal of Computational Intelligence and Applications 5 (2005) 305–320.
[154] J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems), first ed., The MIT Press, 1992.
[155] J.R. Koza, Introduction to genetic programming, in: K.E. Kinnear Jr. (Ed.), Advances in Genetic Programming, MIT Press, Cambridge, MA, USA, 1994, pp. 21–42.
[156] O. Kramer, A review of constraint-handling techniques for evolution strategies, Applied Computational Intelligence and Sof Computing 2010 (2010) 3:1–3:19.
[157] P.V. Laarhoven, E. Aarts, Simulated Annealing: Theory and Applications, first ed., D. Reidel Publishing Company, 1987.
[158] M. Laguna, R. Martí, Scatter Search: Methodology and Implementations in C, Kluwer Academic Publishers, Norwell, MA, USA, 2002.
[159] P. Larrañaga, J.A. Lozano (Eds.), Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation, Kluwer Academic Publishers, Boston, MA, 2002.
[160] L.T. Lau, Guided Genetic Algorithm, Ph.D thesis, University of Essex, 1999.
[161] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (Eds.), The Traveling Salesman Problem, John Wiley & Sons Ltd., Chichester, 1985.
[162] C.Y. Lee, X. Yao, Evolutionary programming using mutations based on the levy probability distribution, IEEE Transactions on Evolutionary Computation 8 (2004) 1–13.
[163] N. Lemmens, S. Jong, K. Tuyls, A. Nowe, A bee algorithm for multi-agent systems: recruitment and navigation combined, in: Proceedings of ALAG 2007, an AAMAS 2007 Workshop, Honolulu, Hawai, USA, 2007.

[164] L. Liberti, M. Drazic, Variable neighbourhood search for the global optimization of constrained NLPS, in: Proceedings of the Global Optimization Workshop, Almeria, Spain, 2005, pp. 18–22.
[165] C.J. Lin, C.H. Chen, C.T. Lin, A hybrid of cooperative particle swarm optimization and cultural algorithm for neural fuzzy networks and its prediction applications, IEEE Transactions on Systems, Man, and Cybernetics – Part C 39 (2009) 55–68.
[166] J. Liu, J. Lampinen, A fuzzy adaptive differential evolution algorithm, Soft Computing 9 (2005) 448–462.
[167] Y. Liu, K. Passino, Biomimicry of social foraging bacteria for distributed optimization: models, principles, and emergent behaviors, Journal of Optimization Theory and Applications 115 (2002) 603–628.
[168] H.R. Lourenço, O. Martin, T. Stützle, Chapter 12: iterated local search: framework and applications, in: M. Gendreau, Y. Potvin (Eds.), Handbook of Metaheuristics, second ed., Springer, 2010, pp. 363–397.
[169] P. Lucic, D. Teodorovic, Computing with bees: attacking complex transportation engineering problems, International Journal on Artificial Intelligence Tools 12 (2003) 375–394.
[170] S. Luke, L. Panait, A survey and comparison of tree generation algorithms, in: L. Spector, E.D. Goodman, A. Wu, W.B. Langdon, H.M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M.H. Garzon, E. Burke (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), Morgan Kaufmann, San Francisco, California, USA, 2001, pp. 81–88.
[171] S. Luke, P.R. Wiegand, When coevolutionary algorithms exhibit evolutionary dynamics, in: A.M. Barry (Ed.), GECCO 2002: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference, AAAI, New York, 2002, pp. 236–241.
[172] H. Ma, An analysis of the equilibrium of migration models for biogeography-based optimization, Information Sciences 180 (2010) 3444–3464.
[173] R. MacArthur, E. Wilson, The Theory of Biogeography, Princeton University Press, Princeton, NJ, 1967.
[174] R. Martí, J.J. Pantrigo, A. Duarte, V. Campos, F. Glover, Scatter search and path relinking: a tutorial on the linear arrangement problem, IJSIR 2 (2011) 1–21.
[175] O. Martin, S.W. Otto, Combining simulated annealing with local search heuristics, Annals of Operations Research 63 (1996) 57–75.
[176] O. Martin, S.W. Otto, E.W. Felten, Large-step Markov chains for the traveling salesman problems, Complex Systems 5 (1991) 299–326.
[177] R.I. McKay, N.X. Hoai, P.A. Whigham, Y. Shan, M. O'Neill, Grammar-based genetic programming: a survey, Genetic Programming and Evolvable Machines 11 (2010) 365–396.
[178] R. Menzel, R.J. De Marco, U. Greggers, Spatial memory, navigation and dance behaviour in apis mellifera, Journal of Comparative Physiology. A – Neuroethology, Sensory, Neural, and Behavioral Physiology 192 (2006) 889–903.
[179] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, Equation of state calculations by fast computing machines, Journal of Chemical Physics 21 (1953) 1087–1090.
[180] S. Meyer-Nieberg, H.G. Beyer, Self-adaptation in evolutionary algorithms, in: F. Lobo, C. Lima, Z. Michalewicz (Eds.), Parameter Setting in Evolutionary Algorithms, Springer, 2007, pp. 47–75.
[181] E. Mezura-Montes, M. Reyes-Sierra, C. Coello Coello, Multi-objective optimization using differential evolution: a survey of the state-of-the-art, in: U. Chakraborty (Ed.), Advances in Differential Evolution, Studies in Computational Intelligence, vol. 143, Springer, Berlin, 2008, pp. 173–196.
[182] P. Mills, Extensions to Guided Local Search, PhD thesis, University of Essex, 2002.
[183] P. Mills, E. Tsang, J. Ford, Applying an extended guided local search to the quadratic assignment problem, Annals of Operations Research 118 (2003) 121–135.
[184] N. Mladenovic, A variable neighborhood algorithm – a new metaheuristic for combinatorial optimization, in: Abstracts of Papers Presented at Optimization Days, Montréal, Canada, 1995, p. 112.
[185] N. Mladenovic, M. Drazic, V. Kovacevic-Vujcic, M. Cangalovic, General variable neighborhood search for the continuous optimization, European Journal of Operational Research 191 (2008) 753–770.
[186] N. Mladenovic, P. Hansen, Variable neighborhood search, Computers and Operations Research 24 (1997) 1097–1100.
[187] O. Montiel, O. Castillo, P. Melin, A. Rodríguez-Díaz, R. Sepúlveda, Human evolutionary model: a new approach to optimization, Information Sciences 177 (2007) 2075–2098.
[188] H. Mühlenbein, The equation for response to selection and its use for prediction, Evolutionary Computation 5 (1997) 303–346.
[189] H. Mühlenbein, T. Mahnig, The factorized distribution algorithm for additively decomposed functions, in: A. Ochoa, M.R. Soto, R. Santana (Eds.), Proceedings of the Second Symposium on Artificial Intelligence (CIMAF-99), Habana, Cuba, 1999, pp. 301–313.
[190] H. Mühlenbein, G. Paaß, From recombination of genes to the estimation of distributions I. Binary parameters, in: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature, PPSN IV, Springer-Verlag, London, UK, 1996, pp. 178–187.
[191] A. Nakib, H. Oulhadj, P. Siarry, Microscopic image segmentation with two-dimensional exponential entropy based on hybrid microcanonical annealing, in: IAPR Conference on Machine Vision Applications, Tokyo, Japan, 2007, pp. 420–423.
[192] S. Nakrani, C. Tovey, On honey bees and dynamic server allocation in internet hosting centers, Adaptive Behavior – Animals, Animats, Software Agents, Robots, Adaptive Systems 12 (2004) 223–240.
[193] J. Nelder, R. Mead, A simplex method for function minimization, The Computer Journal 7 (1965) 308–313.
[194] F. Neri, V. Tirronen, Recent advances in differential evolution: a survey and experimental analysis, Artificial Intelligence Review 33 (2010) 61–106.
[195] T. Nguyen, X. Yao, Hybridizing cultural algorithms and local search, in: E. Corchado, H. Yin, V. Botti, C. Fyfe (Eds.), Intelligent data engineering and automated learning – IDEAL 2006, Lecture Notes in Computer Science, vol. 4224, Springer, Berlin, Heidelberg, 2006, pp. 586–594.
[196] A. Ochoa, J. Ponce, A. Hernández, L. Li, Resolution of a combinatorial problem using cultural algorithms, JCP 4 (2009) 738–741.
[197] A. Ostermeier, A. Gawelczyk, N. Hansen, A derandomized approach to self-adaptation of evolution strategies, Evolutionary Computation 2 (1994) 369–380.
[198] E. Ozcan, C.K. Mohan, Particle swarm optimization: surfing the waves, in: Proceedings of the IEEE Congress on Evolutionary Computation – CEC 1999, 1999, pp. 1939–1944.
[199] M. Pant, R. Thangaraj, A. Abraham, Particle swarm optimization: performance tuning and empirical analysis, in: A. Abraham, A. Hassanien, P. Siarry, A. Engelbrecht (Eds.), Foundations of Computational Intelligence, vol. 3, Springer-Verlag, Berlin, Heidelberg, New York, NY, USA, 2009, pp. 101–128.
[200] K.M. Passino, Biomimicry of bacterial foraging for distributed optimization and control, IEEE Control Systems Magazine 22 (2002) 52–67.
[201] K.M. Passino, Bacterial foraging optimization, International Journal of Swarm Intelligence Research 1 (2010) 1–16.
[202] M. Pelikan, D.E. Goldberg, E. Cant-Paz, BOA: the Bayesian optimization algorithm, Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99 (Orlando, FL), vol. I, Morgan-Kaufmann Publishers, San Fransisco, CA, 1999, pp. 525–532.
[203] M. Pelikan, D.E. Goldberg, E. Cantú-Paz, Linkage problem, distribution estimation, and bayesian networks, Evolutionary Computation 8 (2000) 311–340.
[204] M. Pelikan, D.E. Goldberg, F.G. Lobo, A survey of optimization by building and using probabilistic models, Computational Optimization and Applications 21 (2002) 5–20.
[205] M. Pelikan, H. Mühlenbein, The bivariate marginal distribution algorithm, in: R. Roy, T. Furuhashi, P.K. Chawdhry (Eds.), Advances in Soft Computing – Engineering Design and Manufacturing, Springer-Verlag, London, 1999, pp. 521–535.
[206] A. Piszcz, T. Soule, A survey of mutation techniques in genetic programming, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO '06, ACM, New York, NY, USA, 2006, pp. 951–952.
[207] R. Poli, Analysis of the publications on the applications of particle swarm optimisation, Journal of Artificial Evolution and Applications 2010 (2008) 1–10.
[208] R. Poli, J. Kennedy, T. Blackwell, Particle swarm optimization. An overview, Swarm Intelligence 1 (2007) 33–57.
[209] R. Poli, W.B. Langdon, N.F. McPhee, A Field Guide to Genetic Programming, Lulu Enterprises, UK Ltd., 2008.
[210] J.B. Pollack, A.D. Blair, Co-evolution in the successful learning of backgammon strategy, Machine Learning 32 (1998) 225–240.

[211] E. Popovici, K. De Jong, The effects of interaction frequency on the optimization performance of cooperative coevolution, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO '06, ACM, New York, NY, USA, 2006, pp. 353–360.
[212] M.A. Potter, K.A. De Jong, Cooperative coevolution: an architecture for evolving coadapted subcomponents, Evolutionary Computation 8 (2000) 1–29.
[213] M.A. Potter, K.A.D. Jong, A cooperative coevolutionary approach to function optimization, in: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature, PPSN III, Springer-Verlag, London, UK, 1994, pp. 249–257.
[214] M. Prais, C.C. Ribeiro, Reactive GRASP: an application to a matrix decomposition problem in TDMA traffic assignment, INFORMS Journal on Computing 12 (2000) 164–176.
[215] K.V. Price, R.M. Storn, J.A. Lampinen, Differential Evolution A Practical Approach to Global Optimization, Natural Computing Series, Springer-Verlag, Berlin, Germany, 2005.
[216] I. Rechenberg, Cybernetic Solution Path of an Experimental Problem, Technical Report, Royal Air Force Establishment, 1965.
[217] I. Rechenberg, Evolutionsstrategie: Optimierung Technischer Systeme Nach Prinzipien der Biologischen Evolution, Frommann-Holzboog, Stuttgart, 1973.
[218] A. Renfrew, Dynamic modeling in archaeology: what, when, and where? Dynamical Modeling and the Study of Change in Archaelogy (1994).
[219] M. Resende, C. Ribeiro, R. Martí, F. Glover, Scatter search and path relinking: advances and applications, in: M. Gendreau, J.Y. Potvin (Eds.), Handbook of Metaheuristics, second ed., Springer, 2010, pp. 87–107.
[220] M.G.C. Resende, Metaheuristic hybridization with greedy randomized adaptive search procedures, in: Z.L. Chen, S. Raghavan (Eds.), Tutorials in Operations Research, INFORMS, 2008, pp. 295–319.
[221] M.G.C. Resende, C.C. Ribeiro, Chapter 8: Greedy randomized adaptive search procedures, in: F. Glover, G.A. Kochenberger (Eds.), Handbook of Metaheuristics, Kluwer Academic Publishers, 2003, pp. 219–249.
[222] M.G.C. Resende, C.C. Ribeiro, GRASP: Greedy Randomized Adaptive Search Procedures, Technical Report SGRASP2010, AT&T Labs Research, 2010.
[223] M. Reyes-Sierra, C.A.C. Coello, Multi-objective particle swarm optimizers: a survey of the state-of-the-art, International Journal of Computational Intelligence Research 2 (2006) 287–308.
[224] R.G. Reynolds, An adaptive computer model of plan collection and early agriculture in the eastern valley of oaxaca, Guila Naquitz: Archaic Foraging and Early Agriculture in Oaxaca, Mexico (1986) 439–500.
[225] R.G. Reynolds, An introduction to cultural algorithms, in: A.V. Sebalk, L.J. Fogel (Eds.), Proceedings of the Third Annual conference on Evolutionary Programming, World Scientific Publishing, River Edge, NJ, 1994, pp. 131–139.
[226] R.G. Reynolds, Cultural Algorithms: Theory and Applications, in: New Ideas in Optimization, McGraw-Hill Ltd., UK, Maidenhead, UK, England, 1999, pp. 367–378.
[227] R.G. Reynolds, T.A. Kohler, Z. Kobti, The effects of generalized reciprocal exchange on the resilience of social networks: an example from the prehispanic mesa verde region, Computational and Mathematical Organization Theory 9 (2003) 227–254.
[228] R.G. Reynolds, D. Liu, Multi-objective cultural algorithms, in: IEEE Congress on Evolutionary Computation, 2011, pp. 1233–1241.
[229] R.G. Reynolds, B. Peng, M.Z. Ali, The role of culture in the emergence of decision-making roles: an example using cultural algorithms, Complexity 13 (2008) 27–42.
[230] D.C. Rivera, R.L. Becerra, A. Coello Coello, Carlos, Cultural algorithms, an alternative heuristic to solve the job shop scheduling problem, Engineering Optimization 39 (2007) 69–85.
[231] C.D. Rosin, R.K. Belew, New methods for competitive coevolution, Evolution Computing 5 (1997) 1–29.
[232] N. Rychtyckyj, R.G. Reynolds, Using cultural algorithms to re-engineer large-scale semantic networks, International Journal of Software Engineering and Knowledge Engineering 15 (2005) 665–694.
[233] S. Sadik, A. Ali, H.F. Ahmad, H. Suguri, Using honey bee teamwork strategy in software agents, in: CSCWD'06: 10th International Conference on Computer Supported Cooperative Work in Design, 2006, pp. 1–6.
[234] S.M. Saleem, Knowledge-Based Solution to Dynamic Optimization Problems using Cultural Algorithms, Ph.D. Thesis, Wayne State University, Detroit, MI, USA, 2001.
[235] A. Salehipour, K. Sörensen, P. Goos, O. Bräysy, Efficient GRASP + VND and GRASP + VNS metaheuristics for the traveling repairman problem, 4OR 9 (2011) 189–209.
[236] R. Santana, P. Larrañaga, J. Lozano, Research topics in discrete estimation of distribution algorithms based on factorizations, Memetic Computing 1 (2009) 35–54.
[237] H.P. Schwefel, Numerical Optimization of Computer Models, John Wiley & Sons Inc., New York, NY, USA, 1981.
[238] H.P. Schwefel, Evolution and Optimum Seeking, Wiley, New York, 1995.
[239] H.P. Schwefel, G. Rudolph, Contemporary evolution strategies, in: Proceedings of the Third European Conference on Advances in Artificial Life, Springer Verlag, London, UK, 1995, pp. 893–907.
[240] M. Sebag, M. Schoenauer, C. Ravise, Inductive learning of mutation step-size in evolutionary parameter optimization, in: Evolutionary Programming, 1997, pp. 247–261.
[241] T.D. Seeley, Honeybee Democracy, Princeton University Press, Princeton, NJ, USA, 2010.
[242] T.D. Seeley, A. Mikheyev, G. Pagano, Dancing bees tune both duration and rate of waggle-run production in relation to nectar-source profitability, Journal of Comparative Physiology A 186 (2000) 813–819.
[243] Y. Shan, R.I. McKay, D. Essam, H.A. Abbass, A survey of probabilistic model building genetic programming, in: Scalable Optimization via Probabilistic Modeling, Studies in Computational Intelligence, vol. 33, Springer, 2006, pp. 121–160.
[244] Y. Shi, R. Eberhart, A modified particle swarm optimizer, in: Proceedings of IEEE International Conference on Evolutionary Computation, IEEE Computer Society, Washington, DC, USA, 1998, pp. 69–73.
[245] Y. Shi, R.C. Eberhart, Empirical study of particle swarm optimization, Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on, in: Proceedings of the 1999 Congress on Evolutionary Computation, CEC 99, vol. 3, 1999, pp. 1945–1949.
[246] Y.j. Shi, H.f. Teng, Z.q. Li, Cooperative co-evolutionary differential evolution for function optimization, in: L. Wang, K. Chen, Y. Ong (Eds.), Advances in Natural Computation, Lecture Notes in Computer Science, vol. 3611, Springer, Berlin, Heidelberg, 2005. pp. 428–428.
[247] D. Simon, Biogeography-based optimization, IEEE Transactions on Evolutionary Computation 12 (2008) 702–713.
[248] K. Sims, Evolving 3D morphology and behavior by competition, Artificial Life 1 (1994) 353–372.
[249] A. Sinha, D. Glodberg, A Survey of Hybrid Genetic and Evolutionary algorithms, Technical Report 2003004, Illinois Genetic Algorithms Laboratory (IlliGAL), 2003.
[250] K. Sörensen, M. Sevaux, P. Schittekat, Adaptive and Multilevel Metaheuristics, Adaptive and Multilevel Metaheuristics, Springer, 2008.
[251] M. Soto, A. Ochoa, S. Acid, L.M. de Campos, Introducing the Polytree Approximation of Distribution Algorithm, in: R.S.E.A. Ochoa, M. Soto (Ed.), Proceedings of the Second International Symposium on Artificial Intelligence, Adaptive Systems (International Conference CIMAF'99), 1999, pp. 360–367.
[252] K.O. Stanley, R. Miikkulainen, Competitive coevolution through evolutionary complexification, Journal of Artificial Intelligence Research 21 (2004) 63–100.
[253] R.M. Storn, K.V. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (1997) 341–359.
[254] T. Stützle, Local Search Algorithms for Combinatorial Problems: Analysis, Improvements, and New Applications, Ph.D. Thesis, Darmstadt University of Technology, 1998.

[255] B. Suman, P. Kumar, A survey of simulated annealing as a tool for single and multiobjective optimization, Journal of the Operational Research Society 57 (2006) 1143–1160.
[256] É.D. Taillard, L.M. Gambardella, M. Gendreau, J.Y. Potvin, Adaptive memory programming: a unified view of metaheuristics, European Journal of Operational Research 135 (2001) 1–16.
[257] N. Tairan, Q. Zhang, Population-based guided local search: some preliminary experimental results, in: Proceedings of the IEEE Congress on Evolutionary Computation, Barcelona, Spain, 2010, pp. 1–5.
[258] E.G. Talbi, Metaheuristics: From Design to Implementation, first ed., Wiley-Blackwell, 2009.
[259] C.M. Tan (Ed.), Simulated Annealing, first ed., IN-TECH Education and Publishing, 2008.
[260] K.C. Tan, Y.J. Yang, C.K. Goh, A distributed cooperative coevolutionary algorithm for multiobjective optimization, IEEE Transactions on Evolutionary Computation 10 (2006) 527–549.
[261] N. Teng, J. Teo, M. Hijazi, A. Hanafi, Self-adaptive population sizing for a tune-free differential evolution, Soft Computing 13 (2009) 709–724.
[262] R. Thangaraj, M. Pant, A. Abraham, P. Bouvry, Particle swarm optimization: hybridization perspectives and experimental illustrations, Applied Mathematics and Computation 217 (2011) 5208–5226.
[263] J. Timmis, P. Andrews, E. Hart, On artificial immune systems and swarm intelligence, Swarm Intelligence 4 (2010) 247–273.
[264] J. Timmis, P. Andrews, N. Owens, E. Clark, An interdisciplinary perspective on artificial immune systems, Evolutionary Intelligence 1 (2008) 5–26.
[265] J. Timmis, A. Hone, T. Stibor, E. Clark, Theoretical advances in artificial immune systems, Theoretical Computer Science 403 (2008) 11–32.
[266] E.B. Tylor, Primitive Culture, vol. 2, seventh ed., New York: Brentano's, [1872]1924.
[267] B.H. Ulutas, S. Kulturel-Konak, A review of clonal selection algorithm and its applications, Artificial Intelligence Review 36 (2011) 117–138.
[268] F. Valdez, P. Melin, O. Castillo, Bio-inspired optimization methods for minimization of complex mathematical functions, in: Proceedings of the 10th Mexican International Conference on Artificial Intelligence (MICAI), Puebla, Mexico, 2011a, pp. 131–142.
[269] F. Valdez, P. Melin, O. Castillo, An improved evolutionary method with fuzzy logic for combining particle swarm optimization and genetic algorithms, Applied Soft Computing 11 (2011) 2625–2632.
[270] F. Vandenbergh, A. Engelbrecht, A study of particle swarm optimization particle trajectories, Information Sciences 176 (2006) 937–971.
[271] J.G. Villegas, C. Prins, C. Prodhon, A.L. Medaglia, N. Velasco, GRASP/VND and multi-start evolutionary local search for the single truck and trailer routing problem with satellite depots, Engineering Applications of Artificial Intelligence 23 (2010) 780–794.
[272] C. Voudouris, Guided Local Search for Combinatorial Optimization Problems, Ph.D Thesis, University of Essex, 1997.
[273] C. Voudouris, Guided local search: an illustrative example in function optimization, BT Technology Journal 16 (1998) 46–50.
[274] C. Voudouris, E. Tsang, Guided local search, European Journal of Operational Research 113 (1999) 469–499.
[275] C. Voudouris, E.P.K. Tsang, A. Alsheddy, Effective application of guided local search, in: J.J. Cochran, L.A. Cox, P. Keskinocak, J.P. Kharoufeh, J.C. Smith (Eds.), Wiley Encyclopedia of Operations Research and Management Science, John Wiley & Sons, 2010.
[276] C. Voudouris, E.P.K. Tsang, A. Alsheddy, Guided local search, in: J.J. Cochran, L.A. Cox, P. Keskinocak, J.P. Kharoufeh, J.C. Smith (Eds.), Wiley Encyclopedia of Operations Research and Management Science, John Wiley & Sons, 2010.
[277] A. Walker, J. Hallam, D. Willshaw, Bee-havior in a mobile robot: the construction of a self-organized cognitive map and its use in robot navigation within a complex, natural environment, in: Proc. ICNN'93, Int. Conf. on Neural Networks, vol. III, IEEE Service Center, Piscataway, NJ, 1993, pp. 1451–1456.
[278] H.F. Wedde, M. Farooq, Y. Zhang, BeeHive: an efficient fault-tolerant routing algorithm inspired by honey bee behavior, in: M. Dorigo, M. Birattari, C. Blum, L.M. Gambardella, F. Mondada, T. Stützle (Eds.), Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS 2004, Brussels, Belgium, September 5–8, 2004, Proceedings, Number 3172 in Lecture Notes in Computer Science, Springer, 2004, pp. 83–94.
[279] R.P. Wiegand, An Analysis of Cooperative Coevolutionary Algorithms, Ph.D. Thesis, George Mason University, Fairfax, VA, USA, 2004. AAI3108645.
[280] B.L. William, P. Riccardo, F.M. Nicholas, R.K. John, Genetic programming: an introduction and tutorial, with a survey of techniques and applications, in: J. Fulcher, L.C. Jain (Eds.), Computational Intelligence: A Compendium, Studies in Computational Intelligence (SCI), vol. 115, Springer-Verlag, 2008, pp. 927–1028.
[281] D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation 1 (1997) 67–82.
[282] C. Wu, N. Zhang, J. Jiang, J. Yang, Y. Liang, Improved bacterial foraging algorithms and their applications to job shop scheduling problems, in: Proceedings of the 8th International Conference on Adaptive and Natural Computing Algorithms, Part I, ICANNGA '07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 562–569.
[283] X.S. Yang, Engineering optimizations via nature-inspired virtual bee algorithms, in: Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach: First International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2005, volume 3562 of Lecture Notes in Computer Science, Springer, 2005, pp. 317–323.
[284] X. Yao, Y. Liu, Fast evolutionary programming, in: Evolutionary Programming, 1996, pp. 451–460.
[285] X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Transactions on Evolutionary Computation 3 (1999) 82–102.
[286] Q. Zhang, J. Sun, E. Tsang, J. Ford, Combination of guided local search and estimation of distribution algorithm for quadratic assignment problem, in: Proceedings of the Genetic and Evolutionary Computation Conference, Chicago, IL, USA, 2003, pp. 42–48.
[287] J. Zheng, Y. Chen, W. Zhang, A survey of artificial immune applications, Artificial Intelligence Review 34 (2010) 19–34.