Overview

Evolutionary Algorithms

Thomas Bartz-Beielstein,[1,*] Jürgen Branke,[2] Jörn Mehnen[3] and Olaf Mersmann[1]

Evolutionary algorithm (EA) is an umbrella term used to describe population-based stochastic direct search algorithms that in some sense mimic natural evolution. Prominent representatives of such algorithms are genetic algorithms, evolution strategies, evolutionary programming, and genetic programming. On the basis of the evolutionary cycle, similarities and differences between these algorithms are described. We briefly discuss how EAs can be adapted to work well in case of multiple objectives, and dynamic or noisy optimization problems. We look at the tuning of algorithms and present some recent developments coming from theory. Finally, typical applications of EAs to real-world problems are shown, with special emphasis on data-mining applications.

How to cite this article:
WIREs Data Mining Knowl Discov 2014, 4:178–195. doi: 10.1002/widm.1124

EVOLUTIONARY ALGORITHMS IN A NUTSHELL

This article provides an overview on the state-of-the-art in *evolutionary algorithms* (EAs). It presents a broad and relatively nontechnical treatment of important EA topics, namely the family of EAs, important issues for their application, EA theory, and available software packages.

Invention and development of the first EA is nowadays attributed to a few pioneers who independently suggested four related approaches.[1]

- Fogel et al.[2] introduced *evolutionary programming* (EP) aiming at evolving finite automata, later at solving numerical optimization problems.
- Holland[3] presented *genetic algorithms* (GAs), using binary strings which were inspired by the genetic code found in natural life, to solve combinatorial problems.
- *Evolution strategies* (ESs) as proposed by [4,5] were motivated by engineering problems and thus mostly used a real-valued representation of solution candidates.
- *Genetic programming* (GP), suggested by Koza[6] emerged in the early 1990s. GP explicitly performs the optimization of computer programs.

Since about the same time, these four techniques are collectively referred to as EAs, building the core of the *evolutionary computation* (EC) field.

EAs are understood as population-based stochastic direct search algorithms that in some sense mimic the natural evolution. Points in the search space are considered as individuals (solution candidates), which form a population. Their fitness value is a number, indicating the quality of the solution. Besides initialization and termination as necessary constituents of every algorithm, EAs can consist of three important components: a set of search operators (usually implemented as 'recombination' and 'mutation'), an imposed control flow (Figure 1), and a representation that maps adequate variables to implementable solution candidates (the so-called 'genotype–phenotype mapping'). A widely accepted definition reads as follows:

> 'EA: collective term for all variants of (probabilistic) optimization and approximation algorithms that are inspired by Darwinian evolution. Optimal states are approximated by successive improvements

*Correspondence to: bartzbeielstein@gmail.com

[1]Department of Computer Science, Cologne University of Applied Sciences, Gummersbach, Germany

[2]Warwick Business School, Warwick University, Coventry, UK

[3]Decision Engineering Centre, Cranfield University, Cranfield, UK

Conflict of interest: The authors have declared no conflicts of interest for this article.

based on the variation-selection-paradigm. Thereby, the variation operators produce genetic diversity and the selection directs the evolutionary search'.[8]

Although different EAs may put different emphasis on the search operators mutation and recombination, their general effects are not in question. Mutation means neighborhood based movement in the search space that includes the exploration of the 'outer space' currently not covered by a population, whereas recombination rearranges existing information and thus focuses on the 'inner space'. Selection is meant to introduce a bias toward better fitness values. It can be applied at two stages: When parents are selected from the population to generate offspring (mating selection), and after new solutions have been created and need to be inserted into the population, competing for survival (environmental selection or survival selection). GAs primarily focus on mating selection, ESs utilize only environmental selection.

A typical EA may contain specific mutation, recombination, or selection operators. The EA may employ these operators in every cycle or call them only with a certain probability. However, the general control flow remains unaffected. Each of the consecutive cycles is termed a *generation*. Concerning the representation, it should be noted that most empirical studies are based on canonical forms, such as binary strings or real-valued vectors, whereas many real-world applications require specialized, problem dependent representations.

Bäck[9] compares GAs, ES, and EP. For an in-depth coverage on the defining components of an EA and their connection to natural evolution, see Eiben & Schoenauer[10] and Eiben & Smith[11]. Beyer et al.[8] provide a very useful glossary which covers the basic definitions. De Jong[12] presents an integrated view.

The remainder of this article is structured as follows. After introducing prominent representatives of EAs, namely EP, GAs, ESs, and GP, the following special topics are discussed: multi-objective optimization, dynamic and stochastic optimization, tuning, theory, and applications. The article concludes with a short list of EA related software.

THE FAMILY OF EAS

Starting with its oldest member, namely EP, the family of EAs is described in the following paragraphs. Although EP, GA, ES, and GP were invented independently and described separately, it is unquestioned that these algorithms are specific instances of the more general class of EAs[12] and that it is nowadays difficult to distinguish these algorithms from each other.[13] Only one differentiation is possible even today: EP algorithms do not use recombination. Today, there is a large set of very sophisticated and problem-specific EA implementations. This article cannot be comprehensive and, thus will cover only the most important EA instantiations.

Evolutionary Programming

EP uses a fixed representation and encoding of the solution candidates, while their numerical parameters are allowed to evolve. The essential steps of the EP approach can be described as follows[14]: (1) generate offspring by mutating the individuals in the current population and (2) select the next generation from the offspring and parent population. These key ideas are similarly used in ES, GP, and GAs. However, while ES often uses deterministic selection, selection is often probabilistic in EP. EP operates on the natural problem representation, thus no genotype–phenotype mapping is used. Contrary to GAs and ES, recombination (crossover) is not used in EP. Only mutation is used as the variation operator.

Algorithm 1: Evolutionary programming algorithm

```
1  population = InitializePopulation(populationSize, problemDimension);
2  evaluatePopulation(population);
3  bestSolution = getBestSolution(population);
4  while testForTermination == false do
5      offspring = ∅;
6      for parent_i ∈ population do
7          offspring_i = mutate(parent_i);
8          offspring = {offspring} ∪ offspring_i;
9      evaluatePopulation(offspring);
10     bestSolution = getBestSolution(offspring,bestSolution);
11     population = {population} ∪ {offspring};
12     population = environmentalSelection(population);
13 Return(bestSolution);
```

Algorithm 1 provides a pseudocode listing of an EP algorithm. The steps from Figure 1 are implemented as follows: first, individuals, which form the population, are randomly generated (line 1). Random *initialization* is probably the simplest initialization method. Other, more problem-specific initialization methods are possible. For example, already known good solutions can be used as seeds (starting points) for the initial population. The *evaluation function* (line 2) assigns a quality measure or fitness value to each individual. If the original problem to be solved is an optimization problem, the term *objective function* is used. *Mutation* is a stochastic variation operator which is applied to one individual (line 7). Before the next round in the evolutionary cycle is started, the *environmental* (or survivor) *selection* is performed (line 12). This removes individuals in order to keep the

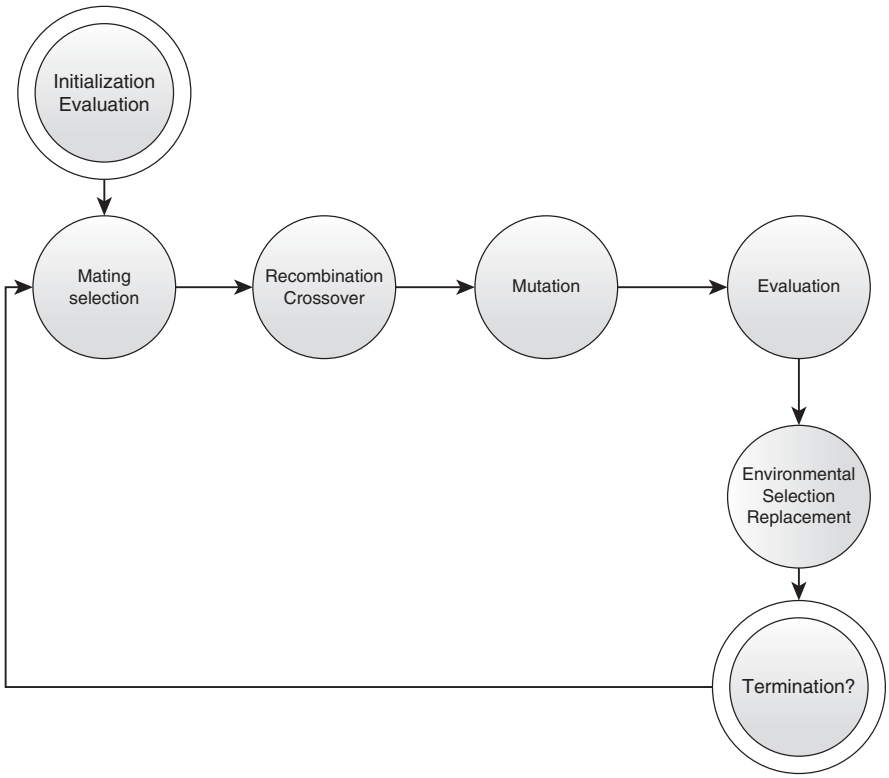

FIGURE 1 | The evolutionary cycle, basic working scheme of all EAs. Common terms for describing ES are used, alternative terms (crossover, replacement) are added below.

population size constant. The decision which individuals to include in the next generation is usually based on fitness values. This evolutionary cycle continues until the *termination* criterion is fulfilled (line 4).

Note, Algorithm 1 provides a simplified template, which can be used as a starting point for elementary EP implementations. This template can be extended in several directions, e.g., by introducing dynamic population sizes.[15]

Fogel[16] summarizes experiences from 40 years of EP, and Fogel and Chellapilla[17] revisit and compare EP with other EA approaches.

Genetic Algorithms

GAs are a variant of EA, which, in analogy to the biological DNA alphabet, originally focused on (bit) string representations. However, alternative encodings have been considered for the representation issue such as real-coded GAs.[18,19]

Binary strings can be decoded in many ways to integer or real values. The string corresponds to the genotype of the individual. The phenotype of the individual is realized by a mapping onto the object parameters, the so-called 'genotype–phenotype mapping'. For example, a binary string that is eight bits long can encode integers between 0 and 255. The genotype '00000011' decodes the integer value 3. The fitness of the individual depends on the optimization problem.

A typical (mating) selection method in GAs is fitness-proportional selection. The probability that an individual is selected for mating depends on its fitness. For example, if a population consists of an individual with fitness value 1 and a second individual with fitness 3, then there is a probability of $1/(1+3) = 1/4$ for selecting the first individual and of $3/(1+3) = 3/4$ of selecting the second individual. Tournament selection is another popular selection method. A tournament is a simple comparison of the fitness values of a small randomly chosen subset of individuals. The individual with the best fitness is the winner of the tournament and will be selected for the recombination (crossover) step. Many other selection methods have been developed for GAs. Any standard EA selection mechanism, e.g., as described in Goldberg[20] can be used.[21]

Mutation adds new information to the population and guarantees that the search process never stops. A simple mutation method is bit-flipping for binary encoded individuals: at a randomly chosen position, a '0' is changed to a '1' and vice versa. For example, the genotype '00000011' can be mutated

to '00000010' if the eighth bit is flipped. Mutation can be performed with a certain probability, which can be decreased during the search. Besides mutation, crossover is an important variation operator for GAs. It has the following two purposes: (1) reduction of the search to more promising regions and (2) inheritance of good gene properties. One-point crossover is a very simple form of crossover. At a randomly chosen position, segments from two individuals are exchanged. Consider two individuals, say '10101010' and '11110000'. One-point crossover at the third position results in two new individuals. The first is '10110000' (with bits one to three from the first parent and bits four to eight from the second parent) and the second is '11110000', respectively. Other popular crossover methods are two-point crossover and uniform crossover.[22] Instead of a fitness-based environmental selection, often a simple generational replacement procedure is used where the newly generated offspring replace their parents independently of fitness.

Algorithm 2: Genetic algorithm

```
1  pop = InitializePopulation(populationSize, problemDimension);
2  evaluatePopulation(population);
3  bestSolution = getBestSolution(population);
4  while testForTermination == false do
5      offspring = ∅;
6      parents = matingSelection(population);
7      for parent_1, parent_2 ∈ parents do
8          (offspring_1,offspring_2) = crossover(parent_1, parent_2);
9          offspring_1 = mutate(offspring_1);
10         offspring_2 = mutate(offspring_2);
11         offspring = {offspring} ∪ offspring_1 ∪ offspring_2;
12     evaluatePopulation(offspring);
13     bestSolution = getBestSolution(offspring);
14     population = replace(population, offspring);
15 Return(bestSolution);
```

Algorithm 2 provides a pseudocode listing of an GA. Note that in addition to the operators used by EP, GAs apply *mating selection* (line 6) and *crossover* (or recombination). Crossover combines information from two or more individuals (line 8). The newly generated offspring replace the parental individuals in the *replacement* procedure (line 14).

Goldberg[20] and Whitley[23] are classical introductions to GAs.

Evolution Strategies

The first ES, the so-called (1 + 1)-ES or two-membered ES, uses one parent and one offspring only. Two rules have been applied to these candidate solutions: Apply small, random changes to all variables simultaneously. If the offspring solution is better (has a better function value) than the parent, take it as the new parent, otherwise retain the parent. Schwefel[24] describes this algorithm as 'the minimal concept for an imitation of organic evolution.' The first (1 + 1)-ES used binomially distributed mutations.[25] These have been replaced by continuous variables and Gaussian mutations, which enable the (1 + 1)-ES to generate larger mutations and thereby possibly escape from local optima. Rechenberg[4] presented an approximate analysis of the (1 + 1)-ES. His analysis showed that the optimal mutation rate corresponds to a success probability that is independent of the problem dimension. The optimal success probability is approximately 1/5 for a linear and also for a quadratic objective function. These results inspired the famous one-fifth rule: Increase the mutation rate if the success rate is larger than 1/5, otherwise, decrease the mutation rate. Schwefel's ES is a variant of EA, which generally operates on the natural problem representation and thus uses no genotype–phenotype mapping for object parameters. In addition to the usual set of decision variables, the individual also contains a set of so-called strategy parameters that influence the mutation operator (e.g., the step size). The ES employs mutation and recombination as variation operators. In general, selection can be performed in two different ways: elitist selection methods include the parental population in the process, whereas nonelitist selection considers the offspring population only. In other words: Elitist methods always keep at least one copy of the fittest solution so far. Elitist and nonelitist selection methods are called plus selection, denoted by $(\mu + \lambda)$, and comma selection, denoted by (μ, λ) in ES, respectively. Other members of the EA family use elitist and nonelitist selection methods as well.[11] Note, unlike selection in GAs, selection in ES is a deterministic procedure. This is similar to animal or plant breeding: only those individuals with promising properties, e.g., high fitness values (objective function values), get a chance of reproduction. Beyer & Schwefel[26] present a comprehensive introduction to ES. Algorithm 3 provides a pseudocode listing of an ES. Please note that the strategy parameters are also subject to recombination (line 8) and mutation (line 9).

The *covariance matrix adaptation evolution strategy* (CMA-ES) is a variant of ES, which was developed for difficult nonlinear nonconvex optimization problems in continuous domains.[27] To motivate the development of the CMA-ES, consider the following black-box optimization problem

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad f(x) \tag{1}$$

and the related problem

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad \widetilde{f}(x) := f(Rx). \tag{2}$$

Algorithm 3: Evolution strategy.

```
population = InitializePopulation(populationSize, problemDimension);
evaluatePopulation(population);
bestSolution = getBestSolution(population);
while testForTermination == false do
    offspring = ∅;
    for i = 0 to offspringSize do
        matingPop = matingSelection(population);
        offspring_i = recombination (matingPop);
        offspring_i = mutate(offspring_i);
        offspring = {offspring} ∪ offspring_i;
    evaluatePopulation(offspring);
    population = environmentalSelection(population);
    bestSolution = getBestSolution(population);
Return(bestSolution);
```

Here $\mathbf{R}$ is an $n \times n$ rotation matrix.[a] Clearly we would like an optimization algorithm to show similar performance characteristics when solving problem (0.1) or (0.2). But neither the $(1+1)$-ES nor Schwefel's ES are *invariant* under rotation and therefore will perform quite differently when solving the two problems. To illustrate this, let us consider a simple example. Let

$$n = 2, \quad f(x) = x_1^2 + 10x_2^2 \quad \text{and}$$
$$R = \begin{pmatrix} \cos(\pi/4) & -\sin(\pi/4) \\ \sin(\pi/4) & \cos(\pi/4) \end{pmatrix}.$$

Here $\mathbf{R}$ is a clockwise rotation of $\mathbf{x} \in \mathbb{R}^n$ around the origin by 45°. Both f and $\tilde{f}$ reach their global minimum in $\mathbf{x}^* = \mathbf{0}$. Their respective function landscape is shown in Figure 2.

The optimal sampling distribution for $f(\mathbf{x})$ should have a covariance structure that (locally) matches the contours of the function landscape. If we restrict ourselves to the multivariate Normal distribution, then the optimal sampling distribution for f around the point $\mathbf{x}$ is given by

$$\mathcal{N}\left(x, \sigma \begin{pmatrix} 1 & 0 \\ 0 & 10 \end{pmatrix}\right).$$

This distribution is clearly covered by both the $(1+1)$-ES and Schwefel's ES. The optimal sampling distribution for $\tilde{f}$ on the other hand is not within the scope of either algorithm:

$$\mathcal{N}\left(x, \sigma R \begin{pmatrix} 1 & 0 \\ 0 & 10 \end{pmatrix}\right).$$

Here, we need to adapt not only just the individual's variances for each parameter but also the covariance structure which models the interdependence between the parameters so that the contour lines of our function and the contour of the search distribution are (locally) similar.

It was this insight that gave rise to the development of the original CMA-ES algorithm.[28] Instead of only adapting the individual variances in each iteration, a full covariance matrix update is performed based on an estimate of the covariance structure from the current population.

In the canonical CMA-ES, offspring are generated by mutating the (sometimes weighted) center of the μ parent individuals, which is usually denoted as $(\mu/\mu, \lambda)$-ES. The strategy parameters include the full covariance matrix instead of just the variance for each dimension of the search space. The update of the strategy parameters is calculated using a maximum-likelihood approach. Here the mean of the search distribution is updated such that the likelihood of selected offspring is maximized. The covariance matrix is incrementally adapted such that the likelihood of successful search steps is maximized.

Because of its richer class of sampling distributions compared to a regular ES and its invariance to rotations of the search space, it is not surprising that the CMA-ES has been very successful at solving both synthetic as well as real-world black-box optimization problems.[29] It is well suited for problems that are nonconvex, nonseparable, ill-conditioned, multi-modal, or if the objective function is noisy. If the objective function is separable, the CMA-ES may not be ideal because it will attempt to learn a covariance structure where there is none to exploit. In such cases a classic ES may outperform the CMA-ES. Recently the update mechanism of the CMA-ES has been recast as a form of natural gradient descent.[30] This has made it possible to adapt the core ideas to other types of continuous search distributions. The tutorial 'Evolution Strategies and CMA-ES'[31] might serve as an introduction to the recent developments in the field of CMA-ES.

Genetic Programming

GP is a collection of EA techniques for the automatic generation of computer programs that perform a user-defined task.[6] Starting with a high-level problem definition, GP creates a population of random programs that are progressively refined through variation and selection until a satisfactory solution is found. GP can be considered as an extension of GAs, because GP does not use fixed-length representations.[32] GP algorithms are applied in various domains such as bioinformatics, data mining, chemical engineering, water industry, and financial trading.[33–36]

One popular GP encoding uses *symbolic expressions* (S-expressions). An S-expression can be defined as (1) an atom, or (2) an expression of the form $(a\ b)$ where a and b are S-expressions. Atoms can be Latin

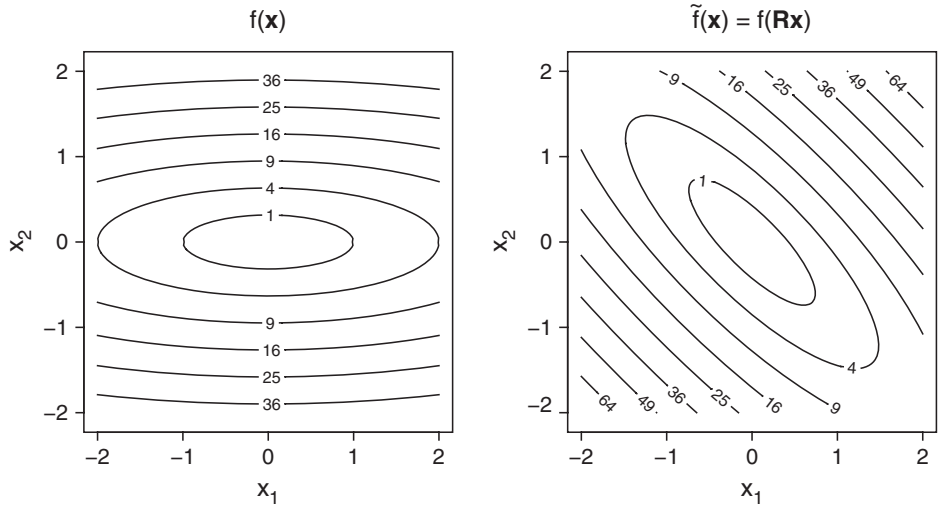

FIGURE 2 | Contour plot of $f(\mathbf{x})$ and $\tilde{f}(x)$ illustrating the effect of rotation on the function landscape.

letters or digits.[37] The expression $(a\ b)$ represents an ordered pair so that S-expressions can be equivalently represented as binary trees. The first element of an S-expression is usually an operator or function name. For example, '$(\sin x)$' and '$(+(12))$' are valid S-expressions in prefix-notation.

After specifying the function and terminal set, an initialization method has to be chosen to generate individuals.[38] Several ways of constructing a collection of random trees are discussed in Poli et al.[21] The *full* method grows binary trees until every branch reaches a pre-specified node depth. The *grow* method generates trees, but no branch is allowed to exceed the pre-specified node depth. The full method tends to create larger trees than the grow method. Since neither one of these two methods is better than the other, the *ramped-half-and-half* initialization proposed by Koza[6] is a very popular tree initialization method. It combines the full and the grow method: Each method is used to created 50% of the individuals. GP allows a variable-length representation, i.e., binary trees with different node depth are members of the same population.

Similar to GAs, the generation of an offspring by combining randomly chosen information from two or more parent programs is referred to as crossover, whereas the creation of an offspring by randomly altering a randomly chosen part of a selected parent program is called mutation. Simple mutation in GP modifies an atom, e.g., replacing '+' with '−'. Subtree mutation is one possible extension, which randomly selects one subtree, which is deleted and regrown in a random manner.

Crossover via subtree exchanging is illustrated in Figure 3. Crossover is applied to two GP individuals,

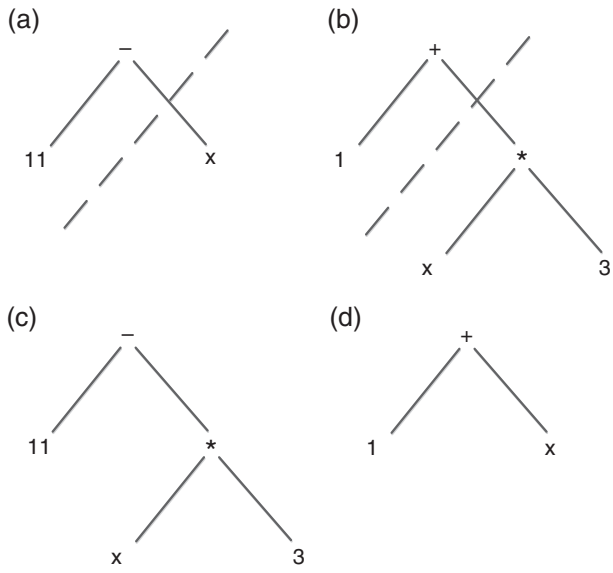

FIGURE 3 | Crossover via subtree exchanging applied to two GP individuals (a) and (b), which are represented as binary trees. The first tree represents the S-expression '$(-(11\ x))$', the second tree represents '$(+(1\ (*\ (x\ 3))))$'. The dashed lines denote the positions, where crossover takes place. Two offspring are created: (c), which represents '$(-(11\ (*\ (x\ 3))))$' and (d), which represents the S-expression '$(+(1\ x))$'.

which are represented as binary trees. The first tree represents the S-expression '$(-(11x))$', which is equivalent to the algebraic expression $11-x$, the second tree represents '$(+(1\ (*\ (x\ 3))))$', which is equivalent to $1+3x$. Two offspring are created: '$(-(11\ (*\ (x\ 3))))$' which is equivalent to $11-3x$ and '$(+(1\ x))$', which is equivalent to $1+x$. In addition to mutation and crossover, reproduction and architecture altering operations can be performed.

An important advantage of GP is that no prior knowledge concerning the solution structure is needed, and solutions can be become arbitrarily complex. Another advantage is the representation of solutions in terms of a formal language (symbolic expressions), i.e., in a form accessible to human reasoning. The main drawback of GP is its high computational cost, due to the potentially infinitely large search space of programs. On the other hand, the recent availability of fast multicore systems has enabled the practical exploitation of GP in many real-world application areas. A common problem of GP is the so-called 'bloat', the growth of solution complexity without benefits in quality. Pareto-GP, which uses solution quality, e.g., prediction error, and model complexity, e.g., the sum of all nodes in all subtrees of the tree, applies an efficient strategy for avoiding bloat.[39,40]

Algorithm 4 provides a pseudocode of a GP algorithm. The basic GP algorithm randomly *selects the genetic operation* (line 7) in its main loop (6). One of the following operations is performed with a certain probability: crossover (line 8), mutation (line 12), or reproduction (line 16).

Poli et al.[21] presents a very comprehensive introduction to GP and can be recommend as a first reading. Koza & Poli[32] is a nice GP tutorial. Langdon & Poli[41] is a classical reference. Gustafson[42] is a good starting point, whereas Langdon[43] lists several hundred GP related references.

SPECIAL TOPICS

Evolutionary Multi-objective Optimization

Many real-world applications require the consideration of *multiple objectives* (MO), for example in information retrieval, where precision (fraction of retrieved instances that are relevant) and recall (fraction of relevant instances retrieved) are common performance criteria. Three different approaches to cope with multiple objectives were proposed during the last decades[44]: (1) transforming the original multi-objective problem into a single-objective problem by using a weighted formula, (2) the Pareto approach, which consists of finding as many nondominated solutions as possible and returning the set of nondominated solutions to the user, and (3) the lexicographical approach, where the objectives are ranked in order of priority.

The standard approach to deal with multiple objectives is to somehow reduce the problem to a single-objective problem. Examples include aggregation (e.g., to a weighted sum $f(x) = \sum_i w_i f_i(x)$), and turning all but one of the objectives into constraints ($\min f_1(x)$ s.t. $f_i \leq c_i \, \forall \, i = 2 \ldots n$). However, this puts a heavy burden on the decision maker, and often she is unable to make such a transformation before knowing the alternatives.

If the objectives $f_1, \ldots, f_n$ are conflicting, then there is usually not a single optimal solution, but a set of so-called 'efficient' or 'Pareto optimal' solutions with different trade-offs between the objectives. An important concept in case of multiple objectives is that of Pareto dominance: a solution x dominates another solution y (written as $x \; y$) iff x is better in at least one objective and not worse in all the others. If a solution x is not dominated by any other solution in the search space, it is said to be *Pareto optimal*, the set of Pareto optimal solutions is called Pareto optimal set.

The lexicographical approach assigns different priorities to different objectives, and then focus on optimizing the objectives in their order of priority.[44] A well-known implementation is the AQ18 rule.[45]

In addition to these three approaches, different approaches are possible with EAs. Because EAs work with a population of solutions, they can be used to generate a set of solutions in one run such as an approximation to the Pareto optimal set. That is, rather than the decision maker having to reduce the problem to a single-objective before optimization, optimization is done on the original multi-objective problem, and the decision maker is provided with a set of Pareto optimal alternatives to choose from. While an approximation of the Pareto set could also be obtained by running an algorithm multiple times, with different weights for the objectives or different constraint settings, running a single multi-objective EA is usually much more efficient. This ability to search for a representative set of Pareto optimal solutions is appealing to many researchers and practitioners, and has made *evolutionary multi-objective optimization* (EMO) one of the most active research areas in EC.

All that needs to be changed when moving from a single-objective EA to a multi-objective EA is the selection process and ranking of individuals in the population. If there is only one objective, individuals are naturally ranked according to this objective, and it is clear which individuals are the best and should be selected as parents or survive to the next generation. In case of multiple objectives, it is still necessary to rank the individuals, but it is no longer obvious how to do this, and many different ranking schemes have been developed. The two most popular multi-objective EAs are probably NSGA-II[46] and SMS-EMOA.[47]

Algorithm 4: Genetic programming algorithm. $\| \cdot \|$ denotes the size of a certain set.

```
1  population = InitializePopulation(populationSize);
2  evaluatePopulation(population);
3  bestSolution = getBestSolution(population);
4  while testForTermination == false do
5      offspring = ∅;
6      while ‖ offspring ‖ < ‖ population ‖ do
7          genOp = selectGeneticOperation;
8          if genOp == crossover then
9              (parent1, parent2) = matingSelection(population);
10             (offspring1,offspring2) = crossover(parent1, parent2);
11             offspring = {offspring} ∪ offspring1 ∪ offspring2;
12         if genOp == mutation then
13             parent = matingSelection(population);
14             offspring1 = mutate(parent);
15             offspring = {offspring} ∪ offspring1;
16         if genOp == reproduction then
17             parent = matingSelection(population);
18             offspring = {offspring} ∪ parent;
19     evaluatePopulation(offspring);
20     bestSolution = getBestSolution(offspring,bestSolution);
21     population = replace(population, offspring);
22 Return(bestSolution);
```

The ranking procedure in NSGA-II is called 'nondominated ranking'. The procedure determines all nondominated solutions and assigns them to the first (best) class. Then, it iteratively removes these solutions from the population, again determines all nondominated solutions, and assigns them the next best class, until the population is empty. Within a class, the algorithm gives the highest rank to the extreme solutions in any objective in order to maintain a wide range of solutions. Then, with the aim of producing an even distribution of solutions, individuals in the same class except for the extreme solutions are sorted according to *crowding distance*. The crowding distance is the sum of differences between an individual's left and right neighbor, in each objective, where large distances (i.e., individuals in a less populated area of the Pareto front) are preferred.

SMS-EMOA uses a concept called *hypervolume* (HV), which measures the volume of the dominated portion of the objective space, bounded by a reference point, see Figure 4. HV has become a popular performance measure in EMO, because it combines in one measure the convergence to the Pareto optimal front and a sensible distribution of solutions along the front. SMS-EMOA ranks individuals according to their marginal contribution to the HV. If HV(P) is the HV of population P, the marginal HV of individual i would be calculated as $\mathrm{MHV}(i) = \mathrm{HV}(P) - \mathrm{HV}(P \backslash \{i\})$, where individuals with larger MHV are preferred. In the example of Figure 4, solution B has the largest marginal HV and would be ranked first.

An excellent online repository on EMO is maintained by Coello.[48]

Innovization

Being able to generate the Pareto optimal set of solutions allows new types of analyses. It is now possible to look at the set of generated solutions, and try to uncover underlying design principles. For example, one can ask what Pareto optimal solutions have in common, or which design variables influence the trade-off of objectives, and in what way. This aspect has been promoted and termed 'innovization' (short for innovation through optimization) by Deb & Srinivasan[49] and it seems to be a field of opportunities for Data Mining and Knowledge Discovery.

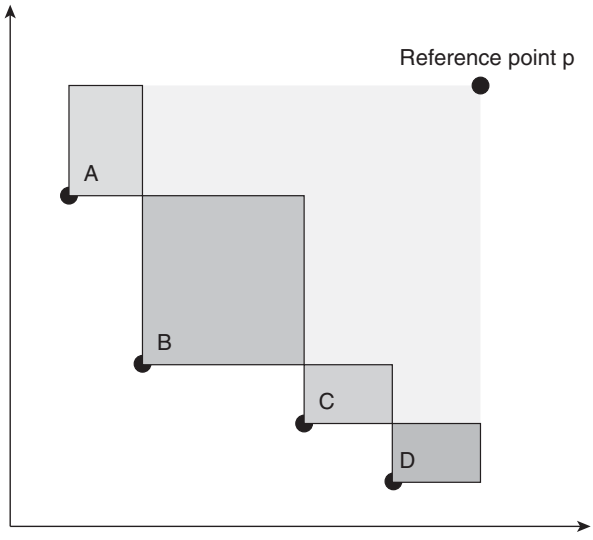

FIGURE 4 | Illustration of (marginal) hypervolume.

Interactive EMO

While it may be impractical for a DM to completely specify his or her preferences before any alternatives are known [and turn the multi-objective (MO) problem into a single-objective (SO) problem as described above], it makes sense to assume that the DMs have at least a rough idea about their preferences, or that partial preference information can be elicited during the optimization by interacting with the DM. This information should be used to speed up the EMO and quickly guide the search toward the solution most preferred by the DM. It also helps in the case of more than three objectives, when the standard EMO algorithms break down because dominance becomes less useful as a criterion for ranking.

Preference information can be elicited in various forms, including, but not limited to, a reference point (desired solution),[50] pairwise comparisons of solutions,[51] or maximum/minimum trade-offs.[52] Recent surveys on integrating user preferences in EMO can be found in.[53,54]

Dynamic Optimization Problems

While most academic papers on optimization deal with fully defined deterministic problems, in the real world, many problems need to deal with uncertainty and varying objectives.

A typical example is dynamic optimization problems, changing over time. In this case, the goal is no longer to find an optimal solution, but to closely track the optimum changing over time. Standard EAs struggle with this task as they tend to converge around the best found solution. When the environment changes, the converged population lacks the diversity required to explore new alternatives and track the optimum. However, a wealth of EA variants has been proposed to deal with this topic. They usually ensure adaptability by maintaining diversity in the population, explicitly increasing diversity after a change has been detected, or dividing the population into several sub-populations to simultaneously explore and track several promising regions in the search space. Recent surveys on this topic can be found in Branke[55] and Jaszkiewicz & Branke[56].

Noisy Environments

Stochastic objective functions, e.g., because evaluation is done through a stochastic simulation or subject to measurement noise, pose special challenges for all optimization algorithms. Such noise makes it difficult even to compare solutions, because the observation that one solution performs better than another could simply be a random effect. However, it has been shown that EAs are relatively insensitive to noise and outperform many other search heuristics on noisy environments.[57,58] Furthermore, EAs can be enhanced, e.g., by integrating statistical re-sampling techniques, the use of surrogate methods to exploit correlation of similar solutions, or adjusting the selection pressure. For a survey of approaches in this area, see Ref 55.

Constraints

When dealing with real-world problem constraints play a major role as almost all real-world problems have to take some kind of limitations of the parameter space into account. EAs can deal with constraints though special techniques, such as the application of penalty functions, decoders, repair mechanisms, constraint preserving operators, or other techniques need to be used. Sometimes it could be worthwhile reformulating a constraint as an objective and vice versa. This technique can be particularly useful in multi-objective optimization where the algorithms are designed to deal with several objectives at the same time. Constraints can also be imposed gradually so that the algorithm may violate constraints in its early explorative stage while getting constrained to the true feasible solution space as the population matures. Although theoretically EAs can find solutions anywhere in the solution space, in reality constraints can direct the algorithms into a suboptimal area having only a very slim chance to escape. Starting the algorithm near a known good solution might help. Also interactive evolution could be a very powerful technique as the user can interactively resolve some issues if the algorithm gets stuck.

Michalewicz & Schoenauer[59] survey several EA based approaches for constrained parameter optimization problems. Coello[60] has compiled a list of more than 1000 references on constraint-handling techniques used with EAs.

Tuning

Many high-performance algorithms—and, in particular, many heuristic solvers, such as EAs for computationally challenging problems, and also many machine learning algorithms—expose parameters to allow end users to adapt the algorithm to target applications. Optimizing parameter settings is thus an important task in the context of developing, evaluating, and applying such algorithms. Recently, a substantial amount of research has been aimed at defining effective procedures for parameter optimization (also called *parameter tuning*).[61–63]

Tuning approaches differ in whether or not explicit models (so-called *response surfaces*) are used to describe the dependence of target algorithm performance on parameter settings. Some notable model-free approaches include F-Race by, Birattari et al.[64] and Balaprakash et al.[65] CALIBRA by Adenso-Diaz & Laguna,[66] and ParamILS by Hutter et al.[67] State-of-the-art model-based approaches use Gaussian stochastic processes (also known as Kriging models) to fit a response surface model. Two independent lines of work extended Kriging to noisy functions, which in the context of parameter optimization, allow the consideration of randomized algorithms: the *sequential Kriging optimization* (SKO) algorithm by Huang et al.[68] and the *sequential parameter optimization* (SPO) procedure by Bartz-Beielstein et al.[69,70]. Eiben & Smit[62] present a comprehensive overview on (off-line) parameter tuning. Wagner[71] discusses the current research on parameter optimization based on experiences from an engineering background.

Parameter control (on-line) is used to change EA parameter values during a run and offers the greatest flexibility and promises the best performance. However, it poses great challenges to EA designers. Eiben et al.[72] serves as a good starting point.

THEORY

The theoretical analysis made some progress over the last decades, but still many open problems are remaining. Rudolph[73] investigated convergence properties of ES. Beyer[13] presents a framework and the first steps toward the theoretical analysis of ES and a recent article by Auger & Hansen[74] presents global convergence results for ES. The *Genetic Programming Theory and Practice* (GPTP) Workshop series discuss the most recent developments in GP theory and practice.[75] Reeves & Rowe[76] and Kallel et al.[77] present theoretical results for GAs. The tutorial slides from Rowe[78] might serve as a good starting point to GA theory.

The existence of a population and the combination of several randomized procedures (mutation, recombination, selection) make EA analysis difficult. Computational complexity theory, which can be seen as the corner stone of computer science, is a popular approach for the theoretical analysis of EAs.[79] In application to randomized search heuristics it takes the form of black-box complexity. Jansen[80] discusses black-box optimization from a complexity-theoretical perspective. However, complexity theory can give paradoxical results such as assigning a low complexity to very hard problems. New definitions, such as parameterized complexity, have been recently proposed.[81] Parameterized complexity classifies computational problems with respect to their number of input parameters. The complexity of a problem is a function in those parameters. Because the complexity of a problem is only measured by the number of bits in the input in classical complexity theory, problems can be classified on a finer scale.

Much of the advances in the theory of EAs has studied simplified algorithms on artificial (toy) problems. The application to real-world problems is much more difficult. An interesting approach is based on landscape analysis. Kauffman & Levin[82] introduced NK fitness landscapes to capture the intuition that both the overall size of the landscape and the number of its local 'hills and valleys' influence the complexity of objective functions. Computing these features can be used to guide the EA search process.

The *no free lunch theorem for search and optimization* (NFL) invoked a heated debate on the performance of optimization algorithms. It can be formulated as follows: all algorithms that search for an extremum of a cost function perform exactly the same when averaged over all possible cost functions.[83] Any improved performance over one class of problems is exactly paid for in performance over another class. Droste et al.[84] argued why the NFL scenario, which applies to finite spaces and algorithms that do not resample points, does not model real life optimization. The reader is guided to Sewell[85] for an overview of NFL and related theorems.

Exploring new methods for designing EAs is also subject of current research.[86,87] And, last but not least, there are many fundamental questions on their working principles for multi-objective optimization problems, which still remain unsolved.[88]

IMPORTANT ISSUES FOR APPLYING EAS

The top four fields of EA applications are in engineering (parameter optimization), medicine, scheduling, and image analysis. The following paragraphs describe important considerations that are necessary for applying EAs in practice.

Choosing the Right Model

Many classical algorithms require simplified problems (e.g., quadratic functions or differentiability) to guarantee exact solutions, whereas EAs generate approximate solutions on the natural problem. EAs are able to work on a model of the real problem, but cannot guarantee convergence to the global optimum.[89]

The design of a fitness function should be concise. Overly detailed functions may use too many parameters which impact negatively on the performance of the search algorithm. As a rule of thumb, maybe a dimension of 30 is typically well manageable by an EA, while any number above 300 may be called high dimensional in evolutionary terms. Linear programming, such as CPLEX, can easily deal with several thousand parameters while being limited to linear problems only. Kordon et al.[90] describe a methodology how to deal with problems in industry. They integrate EAs with statistical methods, neural networks, and support vector machines, and describe applications in the areas of inferential sensors, empirical emulators of mechanistic models, accelerated new product development, complex process optimization, and effective industrial design of experiments.

Expensive Function Evaluations

In industrial applications, superior solutions have been found needing a minimum number of, e.g. 150 fitness function evaluations only. Two approaches, which tackle this problem, are considered next: (1) parallelization and (2) meta-modeling.

Parallel evaluation of several individuals can speed up the algorithm or even increase the probability of finding better solutions through utilizing multiple parallel populations that communicate sporadically through migrants. A discussion of these parallelization concepts goes far beyond the scope of this article, the reader is referred to Cantú-Paz[91] for an elementary introduction. Alba & Tomassini[92] investigate parallelism and EAs. Hu et al.[15] analyze the effect of variable population size on accelerating evolution in the context of a parallel EA. Cantú-Paz[93] reviews parameter settings in parallel GAs.

Meta-modeling can be a very powerful tool in case the evaluation of a fitness function is too expensive. Meta-modeling is a technique that replaces an expensive mathematical model (such as finite element method or computational fluid dynamics) or a complex physical experiment by a often crude but very quick to evaluate model. Statistical approaches such as design of experiments (DoE) from Taguchi type models to sophisticated Kriging are typical. Emmerich[94] describes the development of robust algorithms for optimization with time-consuming evaluations. The main working principle of these techniques is to combine spatial interpolation techniques with EAs. Current results from these demanding real-world applications are presented during the EC in Practice track at the Genetic and Evolutionary Computation Conference (GECCO), see, e.g., http://www.sigevo.org/gecco-2013/ecp.html.

Analyzing the Results

Because the output of an EA run is stochastic, a solution the algorithm finds in one run may slightly or sometimes quite significantly differ from another run. A thorough analysis of EA results needs statistical analysis. Any critical EA analysis should contain at least box-and-whiskers plots to illustrate the statistical spread of the results as any EA run will result typically in a distribution of solutions around any (local) optimum the algorithm finds. Very helpful in industrial applications could be a look at the parameter settings of a solution. A solution close to a constraint could indicate that the optimization problem might be over-constrained or there is a potential better solution in the real-world when a constraint can be relaxed. Also the pattern of the parameter settings in the parameter space can help understanding the underlying problem structure. A random walk structure for example may indicate local plateau areas. In case solutions can be visualized it can sometimes be helpful to view the actual evolution of the solution as a video. This can help improving parameter settings (finding the narrow evolutionary window, i.e., the parameter window where the EA converges best toward better solutions) or determining realistic stopping criteria of the algorithm.

It should be highlighted that robustness of the solution is often a very important criterion in industrial practice. The best solution may be contained in a very narrow set intervals, that when left, the solutions deteriorate quickly. Robust solution in this sense would allow some variation in the parameter settings. Any robustness analysis of a proposed solution gives extra confidence in a result that is supposed to be applied in a critical industrial application. Using the collection of statistical tools from the Analysis of Variance (ANOVA), see, e.g., Ref 95 is highly recommended.

Limitations

Some EAs face limitations when it comes to budgeted fitness evaluations. The stochastic nature of EA may also be a limiting factor when it comes to safety critical applications where repeatability is important. Typically in these cases, EA are used to improve the parameter settings of deterministic algorithms. The explanation of a solution can also be difficult as the way how a solution was deduced is based on a complex stochastic search rather than on a

deterministic one-step-at-a-time approach. A proof that a solution is optimal will not be provided by any current EA, not even how close a solution might be to an optimal solution.

Kordon[96] gives a lively description how to apply EAs in industry and describes several pitfalls. Filipič and Tušar[97] present two case studies of applying optimization methodology in industry, one involving numerical optimization based on simulation models, and the other combinatorial optimization with specific constraints and objectives. They identify some of the challenges frequently met by solution providers for industrial optimization problems.

Data Mining and Knowledge Discovery

EAs have been successfully applied in a large variety of real-world problem areas. Given the focus of the journal, it seems sensible to briefly discuss a particular application area for EAs: data mining and knowledge discovery. In a sense, data mining is about finding good and meaningful models and rules, and thus essentially an optimization task. So it is not surprising that EAs may be helpful also in this area.

They have been proposed for a variety of data-mining related tasks, including feature selection and feature construction, instance selection, or rule extraction. Ghosh & Jain[98] provide a number of examples. Freitas[99–101] presents a comprehensive introduction to data mining and EAs and introduces the term *evolutionary data mining* to subsume any data mining using EAs.

More specific, Tan et al.[102] present a distributed coevolutionary classifier for extracting comprehensible rules in data mining. Vladislavleva et al.[103] forecast the energy output of wind farms using GP and report on the correlation of the different variables for the energy output. Note that GP is able to search for symbolic representations that are interpretable.[104]. Schmidt & Lipson[105] use GP techniques for automatically reverse engineering symbolic analytical models of dynamical systems directly from experimental observations. EAs have been used indirectly to tune parameters of data-mining algorithms[106] or replace machine learning algorithms such as clustering.[107]

Even the automated design of new data-mining algorithms has been proposed.[108] Last, but not least, Schmidt & Lipson[109] has to be mentioned. The authors apply sophisticated GP techniques for the 'identification of nontriviality'. Motion-tracking data captured from various physical systems, e.g., harmonic oscillators and chaotic double-pendula, is used to re-discover Hamiltonians, Lagrangians, and other laws of geometric and momentum conservation. Interestingly, no prior knowledge about physics, kinematics, or geometry, is used by the algorithm.

SOFTWARE

A variety of software frameworks for GP is available: *DataModeler* is a software package that is developed within the context of industrial data analysis.[110] It implements several nonlinear modeling techniques such as Pareto-symbolic regression, statistical learning theory, and nonlinear variable selection. The GP-based software *Discipulus* is applied to Data Mining as well as to problems requiring predictive Analytics and Classification.[111] *Eureqa* is a software tool for detecting equations and hidden mathematical relationships in data.[112,113] *GPTIPS* is a free GP and predictive modeling toolbox for MATLAB.[114]

MATLAB's *Global Optimization Toolbox* has GAs for single and multi objective functions.[115] Implementations of the CMA-ES and links to libraries that contain such implementations can be found on the author's web page[b].[28] The *Java Evolutionary Computation Toolkit* (ECJ) is a freeware EC research system written in Java. It implements several EA techniques, e.g., GAs, GP, and ES.[116] *The MOEA Framework* is an open-source EC library for Java that specializes in multi-objective optimization.[117]

State-of-the-art software packages for parameter tuning and algorithm configuration such as *Bonesa*,[118] *irace*,[119] *ParamILS*,[120] and *SPOT*[121] are freely available from the authors' web pages.

Alcal′a-Fdez et al.[122] develop *KEEL*, an open source Java software tool to assess EAs for data mining problems. Mikut & Reischl[123] discuss the historical development and present a range of existing state-of-the-art data mining and related tools. They provide a list of data-mining tools, which includes EA based approaches, too. Weka and RapidMiner, which provide several machine learning algorithms for solving real-world data-mining problems, contain a couple of EA-based search methods.[124,125] Finally, the statistical software *R*[126] should be mentioned. Several EAs, e.g., *emoa*, *GA*, or *cmaes* are available as R packages, see http://cran.r-project.org/web/packages.

CONCLUSION

EAs are established stochastic direct search algorithms. The evolutionary cycle as depicted in Figure 1 can be seen as the common ground for EA. EAs are trying to reach optimal states by successive improvements. Improvements occur by variation (mutation, recombination). Several problem-specific selection

methods enable to cope with different situations, e.g., noisy and dynamically changing environments. Although EP, GA, ES, and GP stem from different origins, differences between members of this EA family are vanishing. Algorithms learn from each other. For example, Woodward[127] claims that there is 'no reason why the size of a bit string in GAs cannot vary during the evolution. Both crossover and mutation operators, which operate on fixed-length structures, can be engineered into operators which produce variable length bit strings. Conversely, with GP there is no reason why fixed size GP cannot be implemented.' State-of-the-art algorithms combine features from different EA members: Roebber[128] presents an approach in which GP was used to determine a set of if-then equations to describe nonlinear physical processes. Coefficients, operators, and variables are optimized with EP.

Since they are population-based search algorithms, EAs are well suited to solve multi-objective optimization problems. They are also flexible tools for data-mining problems. By modifying the evolutionary cycle, new members of the EA family are generated. Related to GP, *grammatical evolution* (GE) algorithms were proposed.[129] They adopt principles from molecular biology, coupled with the use of grammars to specify legal structures in a search.[130] Bio-inspired algorithms such as *ant colony algorithms*,[131] *particle swarm optimization*,[132] or search heuristics such as differential evolution,[133] enrich the EA family with new problem-specific optimization techniques.

NOTES

[a] A rotation matrix is an orthogonal matrix ($\mathbf{R}^T = \mathbf{R}^{-1}$) with determinant ($\det \mathbf{R} = 1$). The set of all $n \times n$ rotation matrices forms the special orthogonal group $SO(n)$.

[b] https://www.lri.fr/hansen/cmaes_inmatlab.html

REFERENCES

1. Bartz-Beielstein T, Preuß M, Schwefel H-P. Model optimization with evolutionary algorithms. In: Lucas K, Roosen P, eds. *Emergence, Analysis, and Evolution of Structures—Concepts and Strategies Across Disciplines*. Heidelberg, Berlin, New York: Springer; 2010, 47–62.
2. Fogel LJ, Owens AJ, Walsh MJ. Artificial intelligence through a simulation of evolution. In: Callahan A, Maxfield M, Fogel LJ, eds. *Biophysics and Cybernetic Systems*. Washington, DC: Spartan Books; 1965.
3. Holland JH. Genetic algorithms and the optimal allocation of trials. *SIAM J Comput* 1973, 2:88–105.
4. Rechenberg I. Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. PhD Thesis, Department of Process Engineering, Technical University of Berlin, Germany, 1971.
5. Schwefel H-P. *Evolutionsstrategie und numerische Optimierung*. Dr.-Ing. Dissertation. Technische Universität Berlin, Fachbereich Verfahrenstechnik, 1975.
6. Koza J. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press; 1992.
7. Bartz-Beielstein T, Preuss M, Schmitt K, Schwefel H-P. *Challenges for Contemporary Evolutionary Algorithms*. Technical Report TR10-2-003, Faculty of Computer Science, Algorithm Engineering (Ls11), TU Dortmund; 2010.
8. Beyer H-G, Brucherseifer E, Jakob W, Pohlheim H, Sendhoff B, To TB. Evolutionary algorithms—terms and definitions; 2002. Available at: https://homepages.fhv.at/hgb/ea-glossary/ea-terms-engl.html. (Accessed September 3, 2013).
9. Bäck T. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*, vol. 996. Oxford: Oxford University Press; 1996.
10. Eiben AE, Schoenauer M. Evolutionary computing. *Inform Process Lett* 2002, 82:1–6.
11. Eiben AE, Smith JE. *Introduction to Evolutionary Computing*. Heidelberg, Berlin, New York: Springer; 2003.
12. De Jong KA. *Evolutionary Computation: A Unified Approach*, vol. 262041944. Cambridge: MIT Press; 2006.
13. Beyer H-G. *The Theory of Evolution Strategies*. Heidelberg, Berlin, New York: Springer; 2001.
14. Yao X, Liu Y, Lin G. Evolutionary programming made faster. *IEEE Trans Evol Comput* 1999, 3:82–102.
15. Hu T, Harding S, Banzhaf W. Variable population size and evolution acceleration: a case study with a parallel evolutionary algorithm. *Genet Program Evol Mach* 2010, 11:205–225.
16. Fogel LJ. *Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming*. New York: John Wiley & Sons; 1999.
17. Fogel DB, Chellapilla K. Revisiting evolutionary programming. In: *Aerospace/Defense Sensing and Controls*. Bellingham: International Society for Optics and Photonics; 1998, 2–11.

18. Goldberg DE. Real-coded genetic algorithms, virtual alphabets, and blocking. *Urbana* 1990, 51:61801.
19. Herrera F, Lozano M, Verdegay JL. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artif Intell Rev* 1998, 12:265–319.
20. Goldberg DE. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley; 1989.
21. Poli R, Langdon WB, McPhee NF. *A Field Guide to Genetic Programming*, 2008. Published via http://lulu.com and freely available at: http://www.gp-field-guide.org.uk. (Accessed September 13, 2013).
22. Sywerda G. Uniform crossover in genetic algorithms. In: Proceedings of the Third International Conference on Genetic Algorithms. San Francisco, CA: Morgan Kaufmann Publishers Inc, 1989, 2–9.
23. Whitley D. A genetic algorithm tutorial. *Stat Comput* 1994, 4:65–85.
24. Schwefel H-P. *Evolution and Optimum Seeking. Sixth-Generation Computer Technology*. New York: John Wiley & Sons; 1995.
25. Schwefel H-P. *Kybernetische Evolution als Strategie der Exprimentellen Forschung in der Strömungstechnik*. Master's Thesis, Technical University of Berlin, Germany, 1965.
26. Beyer H-G, Schwefel H-P. Evolution strategies: a comprehensive introduction. *Nat Comput* 2002, 1:3–52.
27. Hansen N, Ostermeier A. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: Proceedings of IEEE International Conference on Evolutionary Computation. Piscataway, NJ: IEEE, 1996, 312–317.
28. Hansen N, Ostermeier A, Gawelczyk A. On the adaptation of arbitrary normal mutation distributions in evolution strategies: the generating set adaptation. *ICGA* 1995:57–64.
29. Kern S, Müller S, Hansen N, Büche D, Ocenasek J, Koumoutsakos P. Learning probability distributions in continuous evolutionary algorithms–a comparative review. *Nat Comput* 2004, 3:77–112.
30. Wierstra D, Schaul T, Peters J, Schmidhuber J. Natural evolution strategies. In: IEEE Congress on Evolutionary Computation, 2008. CEC 2008 (IEEE World Congress on Computational Intelligence). Piscataway, NJ: IEEE, 2008, 3381–3387.
31. Auger, A. & Hansen, N. Tutorial: evolution strategies and CMA-ES (covariance matrix adaptation). 2013 Available at: https://www.lri.fr/~hansen/gecco2011-CMA-ES-tutorial.pdf (Accessed September3, 2013).
32. Koza JR, Poli R. A genetic programming tutorial. In: Burke E, Kendall G, eds. *Introductory Tutorials in Optimization, Search and Decision Support* vol. 8, Chapter 8. Heidelberg, Berlin, New York: Springer; 2003.
33. Mercure PK, Smits GF, Kordon A.. Empirical emulators for first principle models. In: AIChE Fall Annual Meeting, 2001, Reno Hilton.
34. Flasch O, Bartz-Beielstein T, Koch P, Konen W. Genetic programming applied to predictive control in environmental engineering. In: Hoffmann F, Hüllermeier E, eds., Proceedings 19. Workshop Computational Intelligence. Karlsruhe: KIT Scientific Publishing, 2009, 101–113.
35. Langdon WB. Large scale bioinformatics data mining with parallel genetic programming on graphics processing units. In: *Parallel and Distributed Computational Intelligence*. Heidelberg, Berlin, New York: Springer; 2010, 113–141.
36. Bacardit J, Llorà X. Large-scale data mining using genetics-based machine learning. *Wiley Interdiscip Rev Data Min Knowl Discov* 2013, 3:37–61.
37. McCarthy J. Recursive functions of symbolic expressions and their computation by machine, part I. *Commun ACM* 1960, 3:184–195.
38. Luke S, Panait L. A survey and comparison of tree generation algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), 2001, 81–88.
39. Laumanns M, Thiele L, Zitzler E, Deb K. Archiving with guaranteed convergence and diversity in multi-objective optimization. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO), 2002, 439–447.
40. Smits GF, Kotanchek M. Pareto-front exploitation in symbolic regression. In: *Genetic Programming Theory and Practice II*. Heidelberg, Berlin, New York: Springer; 2005, 283–299.
41. Langdon WB, Poli R. *Foundations of Genetic Programming*. Heidelberg, Berlin, New York: Springer; 2002.
42. Gustafson S. Steven Gustafson's GP Resources, 2013. Available at: http://www.gustafsonresearch.com/gpresources.html. (Accessed September 4, 2013).
43. Langdon WB. GP home pages, 2013 Available at: http://www0.cs.ucl.ac.uk/staff/W.Langdon/homepags.html. (Accessed September 3, 2013).
44. Freitas AA. A critical review of multi-objective optimization in data mining: a position paper. *SIGKDD Explor Newsl* 2004, 6:77–86.
45. Kaufman KA, Michalski RS. Learning from inconsistent and noisy data: the AQ18 approach. In: Proceedings of the Eleventh International Symposium on Methodologies for Intelligent Systems, 1999, 411–419.
46. Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA–II. *IEEE Trans Evol Comput* 2002, 6:182–197.

47. Beume N, Naujoks B, Emmerich M. SMS-EMOA: multiobjective selection based on dominated hypervolume. *Eur J Oper Res* 2007, 181:1653–1669.
48. Coello CC. (2013). EMOO web page. Available at: http://delta.cs.cinvestav.mx/ccoello/EMOO/. (Accessed September 10, 2013).
49. Deb K, Srinivasan A. Innovization: innovating design principles through optimization. In: Keijzer M, Cattolico M, Arnold D, Babovic V, Blum C, Bosman P, Butz MV, Carlos, Dasgupta D, Ficici SG, et al., eds., Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation. New York, NY: ACM; 2006, 1629–1636.
50. Fonseca CM, Fleming PJ. Multiobjective optimization and multiple constraint handling with evolutionary algorithms – part I: a unified formulation. *IEEE Trans Syst Man Cybern A* 1998, 28:26–37.
51. Branke J, Greco S, Słowiński R, Zielniewicz P. Interactive evolutionary multiobjective optimization using robust ordinal regression. In: Ehrgott M, Fonseca CM, Gandibleux X, Hao J, Marc S, eds., International Conference on Evolutionary Multi-Criterion Optimization, vol. 5467 of *LNCS*. Heidelberg, Berlin, New York: Springer; 2009, 554–568.
52. Branke J, Kaußler T, Schmeck H. Guidance in evolutionary multi-objective optimization. *Adv Eng Softw* 2001, 32:499–507.
53. Branke J. Consideration of user preferences in evolutionary multi-objective optimization. In: Branke J, Deb K, Miettinen K, Slowinski R, eds. *Multiobjective Optimization – Interactive and Evolutionary Approaches*, vol. 5252 of *LNCS*. Heidelberg, Berlin, New York: Springer; 2008, 157–178.
54. Jaszkiewicz A, Branke J. Interactive multi-objective evolutionary algorithms. In: Branke J, Deb K, Miettinen K, Slowinski R, eds. *Multiobjective Optimization – Interactive and Evolutionary Approaches*, vol. 5252 of *LNCS*. Heidelberg, Berlin, New York: Springer; 2008, 179–193.
55. Jin Y, Branke J. Evolutionary optimization in uncertain environments – a survey. *IEEE Trans Evol Comput* 2005, 9:303–317.
56. Nguyen T, Yang S, Branke J. Evolutionary dynamic optimization: a survey of the state of the art. *Swarm Evol Comput* 2012, 6:1–24.
57. Arnold DV. *Noisy Optimization with Evolution Strategies*, vol. 8. Dordrecht: Kluwer Academic Pub; 2002.
58. Arnold DV, Beyer H-G. A comparison of evolution strategies with other direct search methods in the presence of noise. *Comput Optim Appl* 2003, 24: 135–159.
59. Michalewicz Z, Schoenauer M. Evolutionary algorithms for constrained parameter optimization problems. *Evol Comput* 1996, 4:1–32.
60. Coello CAC. List of references on constraint-handling techniques used with evolutionary algorithms. 2013 Available at: http://www.cs.cinvestav.mx/constraint. (Accessed September, 3 2013).
61. Bartz-Beielstein T, Chiarandini M, Paquete L, Preuss M, eds. *Experimental Methods for the Analysis of Optimization Algorithms*. Heidelberg, Berlin, New York: Springer; 2010.
62. Eiben A, Smit S. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm Evol Comput* 2011, 1:19–31.
63. McGeoch CC. *A Guide to Experimental Algorithmics*. 1st ed. New York: Cambridge University Press; 2012.
64. Birattari M, Stützle T, Paquete L, Varrentrapp K. A racing algorithm for configuring metaheuristics. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '02. San Francisco, CA: Morgan Kaufmann Publishers Inc; 2002, 11–18.
65. Balaprakash P, Birattari M, Stützle T. Improvement strategies for the f-race algorithm: sampling design and iterative refinement. In: Proceedings of the 4th International Conference on Hybrid Metaheuristics, HM'07. Heidelberg, Berlin, New York: Springer; 2007, 108–122.
66. Adenso-Diaz B, Laguna M. Fine-tuning of algorithms using fractional experimental design and local search. *Oper Res* 2006, 54:99–114.
67. Hutter F, Hoos HH, Stützle T. Automatic algorithm configuration based on local search. In: AAAI07: Proceedings of the Twenty-Second Conference on Artificial Intelligence; 2007, 1152–1157.
68. Huang D, Allen TT, Notz WI, Zeng N. Global optimization of stochastic black-box systems via sequential kriging meta-models. *J Glob Optim* 2006, 34:441–466.
69. Bartz-Beielstein T, Parsopoulos KE, Vrahatis MN. Design and analysis of optimization algorithms using computational statistics. *Appl Numer Anal Comput Math* 2004, 1:413–433.
70. Bartz-Beielstein T, Lasarczyk C, Preuß M. Sequential parameter optimization. In: Corne D, Michalewicz Z, McKay B, Eiben G, Fogel D, Fonseca C, Greenwood G, Raidl G, Tan KC, Zalzala A, eds., Proceedings 2005 Congress on Evolutionary Computation (CEC'05), Edinburgh, Scotland. Piscataway, NJ: IEEE Press; 2005, vol. 1, 773–780.
71. Wagner T. A subjective review of the state of the art in model-based parameter tuning. In: Workshop on Experimental Methods for the Assessment of Computational Systems (WEMACS 2010), 2010, 1.
72. Eiben A, Hinterding R, Michalewicz Z. Parameter control in evolutionary algorithms. *IEEE Trans Evol Comput* 1999, 3:124–141.

73. Rudolph G. *Convergence Properties of Evolutionary Algorithms*. Hamburg, Germany: Verlag Dr. Kovač; 1997.
74. Auger A, Hansen N. Theory of evolution strategies: a new perspective. In: Auger A, Doerr B, eds. *Theory of Randomized Search Heuristics: Foundations and Recent Developments* Chapter 10. Singapore: World Scientific Publishing; 2011, 289–325.
75. Yu T, Riolo R, Worzel B. *Genetic Programming: Theory and Practice*. Heidelberg, Berlin, New York: Springer; 2006.
76. Reeves CR, Rowe JE. *Genetic Algorithms-Principles and Perspectives: A Guide to GA Theory*, vol. 20. Heidelberg, Berlin, New York: Springer; 2002.
77. Kallel L, Naudts B, Rogers A. *Theoretical Aspects of Evolutionary Computing*. Heidelberg, Berlin, New York: Springer; 2001.
78. Rowe JE. Genetic algorithm theory. In: Soule T, Moore JH, eds. *GECCO (Companion)*. New York, NY: ACM; 2012, 917–940.
79. Wegener I. *Complexity Theory: Exploring the Limits of Efficient Algorithms*. Heidelberg, Berlin, New York: Springer; 2005.
80. Jansen T. *Analyzing Evolutionary Algorithms: The Computer Science Perspective*. Berlin Heidelberg: Springer Publishing Company, Incorporated; 2013.
81. Downey RG, Fellows MR. *Parameterized Complexity*. Heidelberg, Berlin, New York: Springer; 1999.
82. Kauffman S, Levin S. Towards a general theory of adaptive walks on rugged landscapes. *J Theor Biol* 1987, 128:11–45.
83. Wolpert D, Macready W. No free lunch theorems for optimization. *IEEE Trans Evol Comput* 1997, 1:67–82.
84. Droste S, Jansen T, Wegener I. *Perhaps not a free lunch but at least a free appetizer*. Technical Report of the Collaborative Research Centre 531 Computational Intelligence CI-45/98, TU Dortmund; 1998.
85. Sewell M. No free lunch theorems, 2013. Available at: http://www.no-free-lunch.org. (Accessed January 1, 2014).
86. Wierstra D, Schaul T, Glasmachers T, Sun Y, Schmidhuber J. *Natural Evolution Strategies*. Technical Report arxiv:1106.4487v1, arxiv.org, 2011.
87. Rothlauf F. *Design of Modern Heuristics: Principles and Application*. Heidelberg, Berlin, New York: Springer; 2011.
88. Coello CAC, Lamont GB, Veldhuizen DAV. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Secaucus, NJ: Springer-Verlag New York, Inc.; 2006.
89. Michalewicz Z, Fogel DB. *How to Solve it: Modern Heuristics*. Heidelberg, Berlin, New York: Springer; 2004.
90. Kordon, A., Kalos, A., Castillo, F., Jordaan, E., Smits, G., & Kotanchek, M.). Competitive advantages of evolutionary computation for industrial applications. In: The 2005 IEEE Congress on Evolutionary Computation, 2005, vol. 1, pp. 166–173.
91. Cantú-Paz E. Migration policies, selection pressure, and parallel evolutionary algorithms. *J Heuristics* 2001, 7:311–334.
92. Alba E, Tomassini M. Parallelism and evolutionary algorithms. *IEEE Trans Evol Comput* 2002, 6:443–462.
93. Cantú-Paz E. Parameter setting in parallel genetic algorithms. In: *Parameter Setting in Evolutionary Algorithms*. Heidelberg, Berlin, New York: Springer; 2007, 259–276.
94. Emmerich M. *Single- and multi-objective evolutionary design optimization: assisted by Gaussian random field metamodels*. PhD Thesis, Universität Dortmund, Germany, 2005.
95. Montgomery DC. *Design and Analysis of Experiments*. 5th ed. New York: John Wiley & Sons; 2001.
96. Kordon AK. *Applying Computational Intelligence—How to Create Value*. Heidelberg, Berlin, New York: Springer; 2010.
97. Filipiš B, Tušar T. Challenges of applying optimization methodology in industry. In: Proceeding of the Fifteenth Annual Conference Companion on Genetic and Evolutionary Computation Conference Companion, GECCO '13 Companion. New York: ACM; 2013, 1103–1104.
98. Ghosh A, Jain LC, eds. *Evolutionary Computation in Data Mining*. Heidelberg, Berlin, New York: Springer; 2005.
99. Freitas AA. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Heidelberg, Berlin, New York: Springer; 2002.
100. Freitas AA. A survey of evolutionary algorithms for data mining and knowledge discovery. In: Ghosh A, Tsutsui S, eds. *Advances in Evolutionary Computation*. Heidelberg, Berlin, New York: Springer; 2002.
101. Freitas AA. A review of evolutionary algorithms for data mining. In: *Soft Computing for Knowledge Discovery and Data Mining*. Heidelberg, Berlin, New York: Springer; 2008, 79–111.
102. Tan KC, Yu Q, Lee TH. A distributed evolutionary classifier for knowledge discovery in data mining. *IEEE Trans Syst Man Cybern C Appl Rev* 2005, 35:131–142.
103. Vladislavleva E, Friedrich T, Neumann F, Wagner M. Predicting the energy output of wind farms based on weather data: important variables and their correlation. *Renew Energy* 2013, 50:236–243.
104. Vladislavleva E. *Model–based Problem Solving through Symbolic Regression via Pareto Genetic Programming*. PhD thesis, Tilburg University, 2008.

105. Schmidt MD, Lipson H. Data-mining dynamical systems: automated symbolic system identification for exploratory analysis. *ASME Conf Proc* 2008, 2008:643–649.
106. Lessmann S, Stahlbock R., & Crone, S.. Optimizing hyper-parameters of support vector machines by genetic algorithms. In: International Conference on Artificial Intelligence, 2005, 74–82).
107. Handl J, Knowles J. An evolutionary approach to multiobjective clustering. *IEEE Trans Evol Comput* 2007, 11:56–76.
108. Pappa GL, Freitas AA, eds. *Automating the Design of Data Mining Algorithms: An Evolutionary Computation Approach*. Heidelberg, Berlin, New York: Springer; 2010.
109. Schmidt M, Lipson H. Distilling free-form natural laws from experimental data. *Science* 2009, 324:81–85.
110. Evolved Analytics LLC. *DataModeler Release 8.0*. Midland, MI: Evolved Analytics LLC; 2010.
111. Francone FD. *Discipulus—Owner's Manual*. Littleton, CO: Register Machine Learning Technologies, Inc; 2010.
112. Dubčáková R. Eureqa: software review. *Genet Program Evol Mach* 2011, 12:173–178.
113. Austrem PG. A comparative study of the eureqa tool for end-user development. *IJISMD* 2012, 3:66–87.
114. Searson DP, Leahy DE, Willis MJ. GPTIPS: an open source genetic programming toolbox for multigene symbolic regression. In: Proceedings of the International MultiConference of Engineers and Computer Scientists 2010 (IMECS 2010), 2010.
115. Mathworks (2011). *Global Optimization Toolbox Documentation*.
116. Luke S. *The ECJ Owner's Manual—A User Manual for The ECJ Evolutionary Computation Library*. Fairfax, VA: Department of Computer Science, George Mason University; 2013.
117. Hadka D. MOEA framework—a free and open source java framework for multiobjective optimization, 2012. Available at: http://www.moeaframework.org. (Accessed September 2, 2013).
118. Smit S, Eiben AE. Multi-problem parameter tuning using BONESA. In: Hao J, Legrand P, Collet P, Monmarché N, Lutton E, Schoenauer M, eds., Artificial Evolution, 10th International Conference Evolution Artificielle, number 7401 in LNCS. Heidelberg, Berlin, New York: Springer; 2011, 222–233.
119. López-Ibánez M, Dubois-Lacoste J, Stützle T, Birattari M. *The irace package, iterated race for automatic algorithm configuration*. Technical Report TR/IRIDIA/2011-004. IRIDIA, Université Libre de Bruxelles, Belgium, 2011.
120. Hutter F, Bartz-Beielstein T, Hoos H, Leyton-Brown K, Murphy KP. Sequential model-based parameter optimisation: an experimental investigation of automated and interactive approaches. In: Bartz-Beielstein T, Chiarandini M, Paquete L, Preuss M, eds. *Experimental Methods for the Analysis of Optimization Algorithms*. Heidelberg, Berlin, New York: Springer; 2010, 361–414.
121. Bartz-Beielstein T, Zaefferer M. *SPOT Package Vignette*. Technical report, Cologne University of Applied Sciences; 2011.
122. Alcalá-Fdez J, Sánchez L, Garca S, del Jesús MJ, Ventura S, Garrell J, Otero J, Romero C, Bacardit J, Rivas VM, et al. KEEL: a software tool to assess evolutionary algorithms for data mining problems. *Soft Comput* 2009, 13:307–318.
123. Mikut R, Reischl M. Data mining tools. *Wiley Interdiscip Rev Data Min Knowl Discov* 2011, 1:431–443.
124. Witten IH, Frank E. *Data Mining: Practical Machine Learning Tools and Techniques*. Burlington, MA: Morgan Kaufmann; 2005.
125. Rapid-I. *Rapid Miner 5.0 User Manual*. Dortmund: Rapid-I GmbH; 2010.
126. R Core Team. *R: A Language and Environment for Statistical Computing*. Technical report, ISBN 3-900051-07-0. R Foundation for Statistical Computing: Vienna, Austria, 2013; 2005. Available at: http://www.R-project.org. (Accessed September 2, 2013).
127. Woodward J. GA or GP? that is not the question. In: The 2003 Congress on Evolutionary Computation, 2003. CEC '03, vol. 2, 2003, 1056–1063.
128. Roebber PJ. Seeking consensus: a new approach. *Mon Weather Rev* 2010, 138:4402–4415.
129. Ryan C, Collins J, Neill MO. Grammatical evolution: evolving programs for an arbitrary language. In: *Genetic Programming*. Heidelberg, Berlin, New York: Springer; 1998, 83–96.
130. O'Neill M, Ryan C. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*, vol. 4. Heidelberg, Berlin, New York: Springer; 2003.
131. Dorigo M. Optimization, learning and natural algorithms. PhD Thesis, Politecnico di Milano, Italy, 1992.
132. Eberhart, R. & Kennedy, J. A new optimizer using particle swarm theory. In: Proceedings Sixth International Symposium on Micro Machine and Human Science (Nagoya, Japan). Piscataway, NJ: IEEE Service Center, 1995, 39–43.
133. Storn R. On the usage of differential evolution for function optimization. In: Fuzzy Information Processing Society, 1996. NAFIPS, 1996 Biennial Conference of the North American, 1996, 519–523.

FURTHER READING

The *Handbook of Natural Computing* Bäck et al. (2012) describes interactions between computer science and the natural sciences. Brownlee (2011) edited a *Handbook of Algorithmic Recipes*, which contains code for more than forty nature-inspired heuristics.

Bäck T, Kok JN, Rozenberg G. *Handbook of Natural Computing*. Heidelberg, Berlin, New York: Springer; 2012.

Brownlee J. *Clever Algorithms: Nature-Inspired Programming Recipes*. Raleigh, NC: Lulu Enterprises; 2011.

