Article type: Overview

Evolutionary Algorithms[1]

Thomas Bartz-Beielstein

Cologne University of Applied Sciences

Jürgen Branke

Warwick University

Jörn Mehnen

Cranfield University

Olaf Mersmann

Cologne University of Applied Sciences

Keywords

Evolutionary algorithms, Bio-inspired search heuristics, Evolution strategies, Genetic algorithms, Genetic programming

Abstract

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

Evolutionary Algorithms in a Nutshell

Invention and development of the first evolutionary algorithms is nowadays attributed to a few pioneers who independently suggested four related approaches (Bartz-Beielstein et al., 2010b).

- Fogel et al. (1965) introduced *evolutionary programming* (EP) aiming at evolving finite automata, later at solving numerical optimization problems.

[1]This is the pre-peer reviewed version of the following article: Bartz-Beielstein, T. and Branke, J. and Mehnen, J. and Mersmann, O.: Evolutionary Algorithms. WIREs Data Mining Knowl Discov 2014, 4:178-195. doi:10.1002/widm.1124

- Holland (1973) presented *genetic algorithms* (GA), using binary strings, which were inspired by the genetic code found in natural life, to solve combinatorial problems.
- *Evolution strategies* (ES) as proposed by Rechenberg (1971) and Schwefel (1975) were motivated by engineering problems and thus mostly used a real-valued representation.
- *Genetic programming* (GP), suggested by Koza (1992b) emerged in the early 1990s. GP explicitly performs the optimization of programs.

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

Evolutionary algorithms 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 their quality for the problem at hand. Besides initialization and termination as necessary constituents of every algorithm, EAs can consist of three important factors: A set of of search operators (usually implemented as 'recombination' and 'mutation'), an imposed control flow, and a representation that maps adequate variables to implementable solution candidates (the so-called 'genotype-phenotype mapping'). A widely accepted definition reads as follows:

> Evolutionary algorithm: collective term for all variants of (probabilistic) optimization and approximation algorithms that are inspired by Darwinian evolution. Optimal states are approximated by successive improvements based on the variation-selection-paradigm. Thereby, the variation operators produce genetic diversity and the selection directs the evolutionary search (Beyer et al., 2002).

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 so focuses on the 'inner space'. Selection is meant to introduce a bias towards 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 concrete EA may contain specific mutation, recombination, or selection operators, or call them only with a certain probability, but the control flow is usually left unchanged. Each of the consecutive cycles is termed a *generation*.

Concerning the representation, it should be noted that most empiric 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 (1996) 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 (2002) and Eiben & Smith (2003). Beyer et al. (2002) provide a very useful glossary, which covers the basic definitions. De Jong (2006) presents an integrated view.

The remainder of this article is structured as follows. After introducing prominent representatives of EAs, namely evolutionary programming, genetic algorithms, evolution strategies and genetic programming, 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 Evolutionary Algorithms

Starting with its oldest member, namely EP, the family of EAs is described in the following paragraphs. Although EP, GA, ES, and GP have been invented independently and are described separately, it is unquestioned that these algorithms are specific instances of the more general class of EAs (De Jong, 2006) and that it is nowadays difficult to distinguish theses algorithms from each other (Beyer, 2001). Only one differentiation is possible even today: EP algorithms do not use recombination. Today, there is a huge set of very sophisticated and problem-specific EA implementations, and this article can only scotch the surface.

Evolutionary Programming

Evolutionary programming uses a fixed program structure, while its numerical parameters are allowed to evolve. The essential steps of the EP approach can be described as follows (Yao et al., 1999): (i) generate offspring by mutating the individuals in the current population and (ii) 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 used 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 provides a pseudocode listing of an EP algorithm. The steps of an EA are implemented as follows: first, individuals, which form the population, are randomly generated (line 1). Random *initialization* is probably the simplest initialization

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);
```

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) that removes some individuals in order to keep the 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).

Fogel (1999) summarizes experiences from forty years of EP, and Fogel & Chellapilla (1998) revisit and compare EP with other EA approaches.

Genetic Algorithms

Genetic algorithms 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 (Goldberg, 1990; Herrera et al., 1998).

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. Many other selection methods have been developed for GAs, such as tournament selection. A tournament is a simple comparison of the fitness values of a small randomly chosen set of individuals. The individual with the best fitness is the winner of the tournament and will be selected for the recombination (crossover) step.

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: (i) reduction of the search to more promising regions and (ii) 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 (Sywerda, 1989). Instead of a fitness-based environmental selection, often a simple generational replacement procedure is used where the newly generated offspring replaces their parents independent of fitness.

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 replaces the parental individuals in the *replacement* procedure (line 14).

Goldberg (1989) and Whitley (1994) are classical introductions to GAs.

Evolution Strategies

The first ES, the so-called (1+1)-ES or two membered evolution strategy, 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 (1995) describes this algorithm as 'the minimal concept for an imitation of organic evolution.' The first (1+1)-ES used binomially distributed mutations (Schwefel, 1965). 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 (1971) presented an approximate analysis of the (1+1)-ES. His analysis showed that the optimal mutation

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);
```

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 *evolution strategy* (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. Beyer & Schwefel (2002) present a comprehensive introduction to ES. Algorithm 3 provides a pseudocode listing of an ES. Please not 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 non-linear non-convex optimization problems in continuous domains (Hansen & Ostermeier, 1996). To motivate the development of the CMA-ES, consider the following black-box optimization problem

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

and the related problem

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

Here $\mathbf{R}$ is an $n \times n$ rotation matrix[2]. Clearly we would like an optimization algorithm to show similar performance characteristics when solving problem (1) or (2). But neither

[2]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 $SO(n)$.

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);
```

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

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

Here $\mathbf{R}$ is a clockwise rotation of $\mathbf{x} \in \mathbb{R}^n$ around the origin by 45 degrees. Both f and $\tilde{f}$ reach their global minimum in $\mathbf{x}^* = \mathbf{0}$. Their respective function landscape is shown in Figure 1.

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

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

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:

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

Here, we need to adapt not 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 (Hansen et al., 1995). Instead of only adapting the individual variances in each

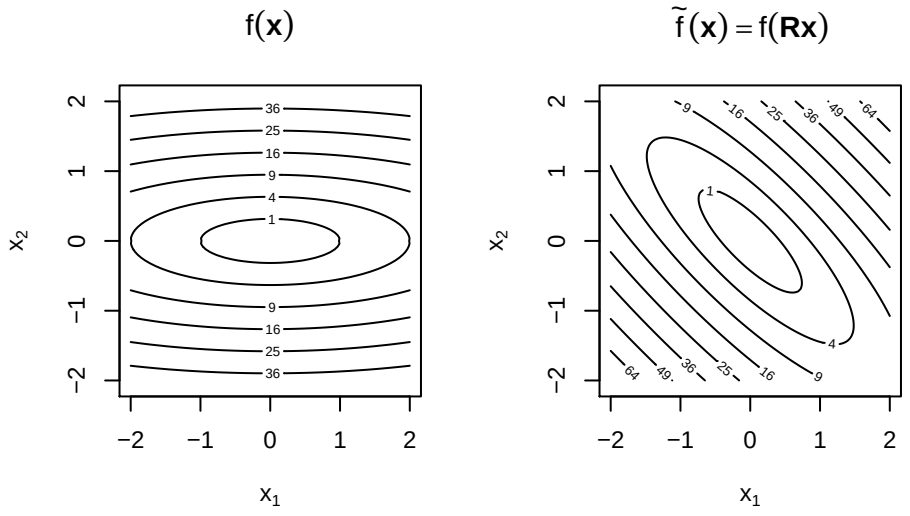

Figure 1: Contour plot of $f(\mathbf{x})$ and $\tilde{f}(\mathbf{x})$ illustrating the effect of rotation on the function landscape.

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 is 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 the mean of the search distribution is updated such that the likelihood of selected offsprings 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 (Kern et al., 2004). It is well suited for problems that are non-convex, non-separable, 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 (Wierstra et al., 2008). This has made it possible to adapt the core ideas to other types of continuous search distributions. The tutorial 'Evolution Strategies and CMA-ES' (Auger & Hansen, 2013) might serve as an introduction to the recent developments in the field of CMA-ES.

Genetic Programming

Genetic programming is a collection of EA techniques for the automatic generation of computer programs that perform a user-defined task (Koza, 1992a). 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 (Koza & Poli, 2003).

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 letters or digits (McCarthy, 1960). 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 '$(+\,(1\,2))$' are valid S-expressions in prefix-notation.

After specifying the function and terminal set, an initialization method has to be chosen to generate individuals (Luke & Panait, 2001). Several ways of constructing a collection of random trees are discussed in Poli et al. (2008). The *ramped-half-and-half* initialization proposed by Koza (1992a) is one popular method. In contrast to GAs and ES, GP allows a variable-length representation, i.e, binary trees with different node depth are member 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 sub-tree exchanging is illustrated in Fig. 2. Crossover is applied to two GP individuals, which are represented as binary trees. The first tree represents the S-expression '$(-\,(11\,x))$', 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 multi-core 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 (Laumanns et al., 2002; Smits & Kotanchek, 2005).

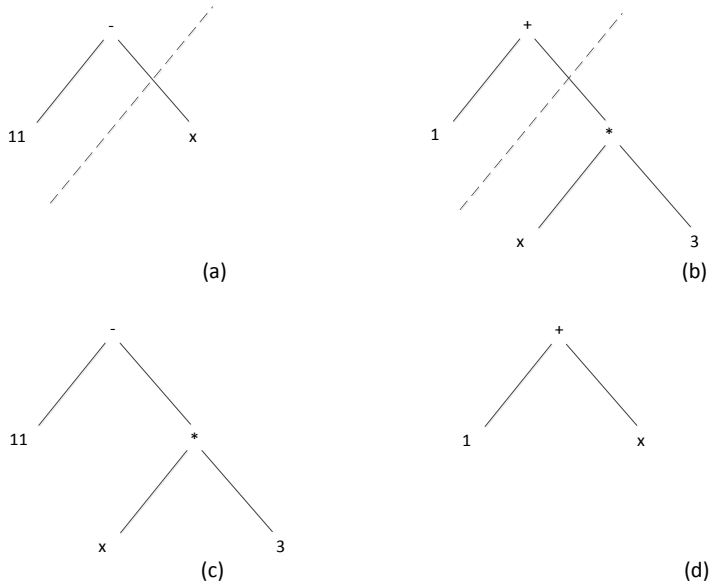

Figure 2: Crossover via sub-tree exchanging applied to two GP individuals (a) and (b), which are represented as binary trees. The first tree represents the S-expression '$(-\,(11\,x))$', the second tree represents '$(+\,(1\,(*\,(x\,3))))$'. The dashed lines denote the positions, where crossover takes place. Two offspring are created: (c), which represents '$(-\,(11\,(*(x\,3))))$' and (d) , which represents the S-expression '$(+\,(1\,x))$'.

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. (2008) presents a very comprehensive introduction to GP and can be recommend as a first reading. Koza & Poli (2003) is a nice GP tutorial.

Special Topics

Evolutionary Multi-objective Optimization (EMO)

Many real-world applications require the consideration of multiple objectives, 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. If the objectives $f_1, \ldots, f_n$ are conflicting, then there is usually not a single optimal solution, but a large set of so-called 'efficient' or 'Pareto optimal' solutions

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              (parent_1, parent_2) = matingSelection(population);
10             (offspring_1,offspring_2) = crossover(parent_1, parent_2);
11             offspring = {offspring} ∪ offspring_1 ∪ offspring_2;
12         if genOp == mutation then
13             parent = matingSelection(population);
14             offspring_1 = mutate(parent);
15             offspring = {offspring} ∪ offspring_1;
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);
```

with different trade-offs of the objectives. An important concept in case of multiple objectives is that of Pareto dominance: a solution x dominates another solution y (written as $x \succ y$) 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 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* (DM), and often the DM is unable to make such a transformation before knowing the alternatives.

A completely different approach is 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 DM having to reduce the problem to a single-objective before optimization, optimization is done on the original multi-objective problem, and the DM 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 how individuals in the population are ranked. 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 (Deb et al., 2002) and SMS-EMOA (Beume et al., 2007).

The ranking procedure in NSGA-II is called 'non-dominated ranking'. The procedure determines all non-dominated solutions and assigns them to the first (best) class. Then, it iteratively removes these solutions from the population, again determines all non-dominated 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 3. 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 hypervolume of population P, the marginal HV of individual i would be calculated as $MHV(i) = HV(P) - HV(P \setminus \{i\})$, where individuals with larger MHV are preferred. In the example of Fig. 3, solution B has the largest marginal hypervolume and would be ranked first.

An excellent on-line repository on EMO is maintained by Coello (2013b)

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, 2006), and it seems to be a field of opportunities for Data Mining and Knowledge Discovery.

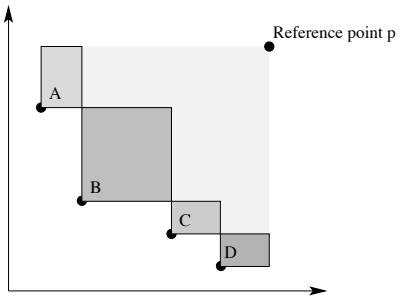

Figure 3: Illustration of (marginal) Hypervolume.

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 towards 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) (Fonseca & Fleming, 1998), pairwise comparisons of solutions (Branke et al., 2009), or maximum/minimum trade-offs (Branke et al., 2001). Recent surveys on integrating user preferences in EMO can be found in Branke (2008); Jaszkiewicz & Branke (2008).

Dynamic and Stochastic 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.

Dynamic Optimization Problems

A typical example are 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 subpopulations to simultaneously explore and track several promising regions in the search space. Recent surveys on this topic can be found in Jin & Branke (2005); Nguyen et al. (2012)

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 (Arnold, 2002; Arnold & Beyer, 2003). 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 Jin & Branke (2005).

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*) (Bartz-Beielstein et al., 2010a; Eiben & Smit, 2011; McGeoch, 2012).

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. (2002); Balaprakash et al. (2007), CALIBRA by Adenso-Diaz & Laguna (2006), and ParamILS by Hutter et al. (2007). 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. (2006), and the *sequential parameter optimization* (SPO) procedure by Bartz-Beielstein et al. (2004, 2005). Eiben & Smit (2011) present a comprehensive overview on (off-line) parameter tuning. Wagner (2010) 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. (1999) 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 (1997) investigated convergence properties of ES. Beyer (2001) presents a framework and the first steps toward the theoretical analysis of ES and a recent article by Auger & Hansen (2011) 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 (Yu et al., 2006). Reeves & Rowe (2002) present theoretical results for GAs. The tutorial slides from Rowe (2012) 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 (Wegener, 2005). In application to randomized search heuristics it takes the form of black-box complexity. Jansen (2013) 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 (Downey & Fellows, 1999). 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 evolutionary algorithms 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 (1987) 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.

Exploring new methods for designing EAs is also subject of current research (Wierstra et al., 2011; Rothlauf, 2011). And, last but not least, there are many fundamental questions on their working principles for multi-objective optimization problems, which still remain unsolved (Coello et al., 2006).

Applications

Analyzing the papers published in the most popular EA conferences (i.e., GECCO, WCCI, PPSN) reveals that 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 (Michalewicz & Fogel, 2004).

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. (2005) 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.

Constraints

When dealing with real-world problems constraints play a major role as almost all real-world problems have to take some kind of limitations of the parameter space into account. Evolutionary algorithms 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 evolutionary algorithms 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 (1996) survey several EA based approaches for constrained parameter optimization problems. Coello (2013a) has compiled a list of more than 1,000 references on constraint-handling techniques used with EAs.

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: (i) parallelization and (ii) 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 (2001) for an elementary introduction. Alba & Tomassini (2002) investigate parallelism and EAs. Hu et al. (2010) analyze the effect of variable population size on accelerating evolution in the context of a parallel EA. Cantú-Paz (2007) 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 FEM or CFD) 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 (2005) 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 GECCO's evolutionary computation in practice trace, 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-constraint 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 through CAD 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 towards to 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 (e.g., using ANOVA) gives extra confidence in a result that is supposed to be applied in a critical industrial application.

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 (2010) gives a lively description how to apply EAs in industry and describes several pitfalls. Filipič & Tušar (2013) 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 (2005) provide a number of examples. Freitas (2002a,b, 2008) 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. (2005) present a distributed coevolutionary classifier for extracting comprehensible rules in data mining. Vladislavleva et al. (2013) 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 (Vladislavleva, 2008). Schmidt & Lipson (2008) 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 (Lessmann et al., 2005) or replace machine learning algorithms such as clustering (Handl & Knowles, 2007).

Even the automated design of new data mining algorithms has been proposed (Pappa & Freitas, 2010). Last, but not least, Schmidt & Lipson (2009) 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 (Evolved Analytics LLC, 2010). It implements several non-linear modeling techniques such as Pareto-symbolic regression, statistical learning theory, and non-linear variable selection. The GP-based software *Discipulus* is applied to Data Mining as well as to problems requiring predictive Analytics and Classification (Francone, 2010). *Eureqa* is a software tool for detecting equations and hidden mathematical relationships in data (Dubčáková, 2011; Austrem, 2012). *GPTIPS* is a free genetic programming (GP) and predictive modeling toolbox for MATLAB (Searson et al., 2010).

MATLAB's *Global Optimization Toolbox* has genetic algorithms for single and multi objective functions (Mathworks, 2011). Implementations of the CMA-ES and links to libraries that contain such implementations can be found on the author's web page[3] (Hansen et al., 1995). The *Java Evolutionary Computation Toolkit* (ECJ) is a freeware evolutionary computation research system written in Java. It implements several EA techniques, e.g., GAs, GP, and ES (Luke, 2013). *The MOEA Framework* is an open-source evolutionary computation library for Java that specializes in multi-objective optimization (Hadka, 2012).

State-of-the-art software packages for parameter tuning and algorithm configuration such as *Bonesa* (Smit & Eiben, 2011), *irace* (López-Ibánez et al., 2011), *ParamILS* (Hutter et al., 2010), and *SPOT* (Bartz-Beielstein & Zaefferer, 2011) are freely available from the authors' web pages.

Alcalá-Fdez et al. (2009) develop *KEEL*, an open source Java software tool to assess evolutionary algorithms for data mining problems. Mikut & Reischl (2011) 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

[3] `https://www.lri.fr/~hansen/cmaes_inmatlab.html`

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 (Witten & Frank, 2005; Rapid-I, 2010). Finally, the statistical software *R* (R Core Team, 2005) 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

Evolutionary algorithms are established stochastic direct search algorithms. The evolutionary cycle can be seen as the common ground for EA. They 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. 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. Bio-inspired algorithms such as *particle swarm optimization* (Eberhart & Kennedy, 1995) or *ant colony algorithms* (Dorigo, 1992) enrich the EA family with new problem specific optimization techniques.

References

Adenso-Diaz, B. & Laguna, M. (2006). Fine-tuning of algorithms using fractional experimental design and local search. *Operations Research*, 54(1), 99–114.

Alba, E. & Tomassini, M. (2002). Parallelism and evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, 6(5), 443–462.

Alcalá-Fdez, J., Sánchez, L., García, S., del Jesús, M. J., Ventura, S., Garrell, J., Otero, J., Romero, C., Bacardit, J., Rivas, V. M., et al. (2009). KEEL: a software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, 13(3), 307–318.

Arnold, D. V. (2002). *Noisy optimization with evolution strategies*, volume 8. Kluwer Academic Pub.

Arnold, D. V. & Beyer, H.-G. (2003). A comparison of evolution strategies with other direct search methods in the presence of noise. *Computational Optimization and Applications*, 24(1), 135–159.

Auger, A. & Hansen, N. (2011). Theory of evolution strategies: a new perspective. In A. Auger & B. Doerr (Eds.), *Theory of Randomized Search Heuristics: Foundations and Recent Developments* chapter 10, (pp. 289–325). World Scientific Publishing.

Auger, A. & Hansen, N. (2013). Evolution strategies and cma-es (covariance matrix adaptation).

Austrem, P. G. (2012). A comparative study of the eureqa tool for end-user development. *IJISMD*, 3(3), 66–87.

Bäck, T. (1996). *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*, volume 996. Oxford university press Oxford.

Bäck, T., Kok, J. N., & Rozenberg, G. (2012). *Handbook of Natural Computing*. Springer.

Balaprakash, P., Birattari, M., & Stützle, T. (2007). Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In *Hybrid Metaheuristics* (pp. 108–122).

Bartz-Beielstein, T., Chiarandini, M., Paquete, L., & Preuss, M., Eds. (2010a). *Experimental Methods for the Analysis of Optimization Algorithms*. Berlin, Heidelberg, New York: Springer.

Bartz-Beielstein, T., Lasarczyk, C., & Preuß, M. (2005). Sequential parameter optimization. In B. McKay & others (Eds.), *Proceedings 2005 Congress on Evolutionary Computation (CEC'05), Edinburgh, Scotland*, volume 1 (pp. 773–780). Piscataway NJ: IEEE Press.

Bartz-Beielstein, T., Parsopoulos, K. E., & Vrahatis, M. N. (2004). Design and analysis of optimization algorithms using computational statistics. *Applied Numerical Analysis and Computational Mathematics (ANACM)*, 1(2), 413–433.

Bartz-Beielstein, T., Preuß, M., & Schwefel, H.-P. (2010b). Model optimization with evolutionary algorithms. In K. Lucas & P. Roosen (Eds.), *Emergence, Analysis, and Evolution of Structures—Concepts and Strategies Across Disciplines* (pp. 47–62). Berlin, Heidelberg, New York: Springer.

Bartz-Beielstein, T. & Zaefferer, M. (2011). *SPOT Package Vignette*. Technical report, Cologne University of Applied Sciences.

Beume, N., Naujoks, B., & Emmerich, M. (2007). SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3), 1653–1669.

Beyer, H.-G. (2001). *The Theory of Evolution Strategies*. Berlin, Heidelberg, New York: Springer.

Beyer, H.-G., Brucherseifer, E., Jakob, W., Pohlheim, H., Sendhoff, B., & To, T. B. (2002). Evolutionary algorithms—terms and definitions. http://ls11-www.cs.uni-dortmund.de/people/beyer/EA-glossary/def-engl-html.html.

Beyer, H.-G. & Schwefel, H.-P. (2002). Evolution strategies: A comprehensive introduction. *Natural Computing*, 1(1), 3–52.

Birattari, M., Stützle, T., Paquete, L., & Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '02 (pp. 11–18). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Branke, J. (2008). Consideration of user preferences in evolutionary multiobjective optimization. In J. Branke, K. Deb, K. Miettinen, & R. Slowinski (Eds.), *Multiobjective Optimization - Interactive and Evolutionary Approaches*, volume 5252 of *LNCS* (pp. 157–178). Springer.

Branke, J., Greco, S., Słowiński, R., & Zielniewicz, P. (2009). Interactive evolutionary multiobjective optimization using robust ordinal regression. In M. Ehrgott & others (Eds.), *International Conference on Evolutionary Multi-Criterion Optimization*, volume 5467 of *LNCS* (pp. 554–568).: Springer.

Branke, J., Kaußler, T., & Schmeck, H. (2001). Guidance in evolutionary multiobjective optimization. *Advances in Engineering Software*, 32, 499–507.

Brownlee, J. (2011). *Clever algorithms: nature–inspired programming recipes*. Lulu Enterprises.

Cantú-Paz, E. (2001). Migration policies, selection pressure, and parallel evolutionary algorithms. *Journal of Heuristics*, 7(4), 311–334.

Cantú-Paz, E. (2007). Parameter setting in parallel genetic algorithms. In *Parameter Setting in Evolutionary Algorithms* (pp. 259–276). Springer.

Coello, C. A. C. (2013a). List of references on constraint-handling techniques used with evolutionary algorithms. http://www.cs.cinvestav.mx/ constraint/.

Coello, C. A. C., Lamont, G. B., & Veldhuizen, D. A. V. (2006). *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Coello, C. C. (2013b). Emoo web page. `http://delta.cs.cinvestav.mx/~ccoello/EMOO/`.

De Jong, K. A. (2006). *Evolutionary computation: a unified approach*, volume 262041944. MIT press Cambridge.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. *IEEE Transactions on Evolutionary Computation*, 6(2), 182–197.

Deb, K. & Srinivasan, A. (2006). Innovization: Innovating design principles through optimization. In M. Keijzer & others (Eds.), *Proceedings of the 8th annual conference on Genetic and evolutionary computation* (pp. 1629–1636).: ACM.

Dorigo, M. (1992). Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano, Italy*.

Downey, R. G. & Fellows, M. R. (1999). *Parameterized Complexity*. Springer-Verlag.

Dubčáková, R. (2011). Eureqa: software review. *Genetic Programming and Evolvable Machines*, 12(2), 173–178.

Eberhart, R. & Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Proceedings Sixth International Symposium on Micro Machine and Human Science (Nagoya, Japan)* (pp. 39–43). Piscataway NJ: IEEE Service Center.

Eiben, A., Hinterding, R., & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2), 124–141.

Eiben, A. & Smit, S. (2011). Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1), 19 – 31.

Eiben, A. E. & Schoenauer, M. (2002). Evolutionary computing. *Information Processing Letters*, 82(1), 1–6.

Eiben, A. E. & Smith, J. E. (2003). *Introduction to Evolutionary Computing*. Berlin, Heidelberg: Springer.

Emmerich, M. (2005). *Single- and Multi-objective Evolutionary Design Optimization: Assisted by Gaussian Random Field Metamodels*. PhD thesis, Universität Dortmund, Germany.

Evolved Analytics LLC (2010). *DataModeler Release 8.0*. Evolved Analytics LLC.

Filipič, B. & Tušar, T. (2013). 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 (pp. 1103–1104). New York, NY, USA: ACM.

Fogel, D. B. & Chellapilla, K. (1998). Revisiting evolutionary programming. In *Aerospace/Defense Sensing and Controls* (pp. 2–11).: International Society for Optics and Photonics.

Fogel, L. J. (1999). *Intelligence through simulated evolution: forty years of evolutionary programming*. John Wiley & Sons, Inc.

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1965). Artificial intelligence through a simulation of evolution. In A. Callahan, M. Maxfield, & L. J. Fogel (Eds.), *Biophysics and Cybernetic Systems*. Washington DC: Spartan Books.

Fonseca, C. M. & Fleming, P. J. (1998). Multiobjective optimization and multiple constraint handling with evolutionary algorithms - part I: A unified fomulation. *IEEE Transactions on Systems, Man, and Cybernetics - Part A*, 28(1), 26–37.

Francone, F. D. (2010). *Discipulus—Owner's Manual*. Register Machine Learning Technologies, Inc.

Freitas, A. A. (2002a). *Data mining and knowledge discovery with evolutionary algorithms*. Springer.

Freitas, A. A. (2002b). A survey of evolutionary algorithms for data mining and knowledge discovery. In *In: A. Ghosh, and S. Tsutsui (Eds.) Advances in Evolutionary Computation*: Springer-Verlag.

Freitas, A. A. (2008). A review of evolutionary algorithms for data mining. In *Soft Computing for Knowledge Discovery and Data Mining* (pp. 79–111). Springer.

Ghosh, A. & Jain, L. C., Eds. (2005). *Evolutionary Computation in Data Mining*. Springer.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading MA: Addison-Wesley.

Goldberg, D. E. (1990). Real-coded genetic algorithms, virtual alphabets, and blocking. *Urbana*, 51, 61801.

Hadka, D. (2012). *MOEA Framework—A Free and Open Source Java Framework for Multiobjective Optimization*.

Handl, J. & Knowles, J. (2007). An evolutionary approach to multiobjective clustering. *IEEE Transactions*, 11(1), 56–76.

Hansen, N. & Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on* (pp. 312–317).: IEEE.

Hansen, N., Ostermeier, A., & Gawelczyk, A. (1995). On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation. In *ICGA* (pp. 57–64).

Herrera, F., Lozano, M., & Verdegay, J. L. (1998). Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artif. Intell. Rev.*, 12(4), 265–319.

Holland, J. H. (1973). Genetic algorithms and the optimal allocation of trials. *SIAM Journal of Computing*, 2(2), 88–105.

Hu, T., Harding, S., & Banzhaf, W. (2010). Variable population size and evolution acceleration: a case study with a parallel evolutionary algorithm. *Genetic Programming and Evolvable Machines*, 11(2), 205–225.

Huang, D., Allen, T. T., Notz, W. I., & Zeng, N. (2006). Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of Global Optimization*, 34(3), 441–466.

Hutter, F., Bartz-Beielstein, T., Hoos, H., Leyton-Brown, K., & Murphy, K. P. (2010). Sequential model-based parameter optimisation: an experimental investigation of automated and interactive approaches. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, & M. Preuss (Eds.), *Experimental Methods for the Analysis of Optimization Algorithms* (pp. 361–414). Berlin, Heidelberg, New York: Springer.

Hutter, F., Hoos, H. H., & Stützle, T. (2007). Automatic algorithm configuration based on local search. In *IN AAAI07: PROC. OF THE TWENTY-SECOND CONFERENCE ON ARTIFICAL INTELLIGENCE* (pp. 1152–1157).

Jansen, T. (2013). *Analyzing Evolutionary Algorithms: The Computer Science Perspective*. Springer Publishing Company, Incorporated.

Jaszkiewicz, A. & Branke, J. (2008). Interactive multi-objective evolutionary algorithms. In J. Branke, K. Deb, K. Miettinen, & R. Slowinski (Eds.), *Multiobjective Optimization - Interactive and Evolutionary Approaches*, volume 5252 of *LNCS* (pp. 179–193). Springer.

Jin, Y. & Branke, J. (2005). Evolutionary optimization in uncertain environments – a survey. *IEEE Transactions on Evolutionary Computation*, 9(3), 303–317.

Kauffman, S. & Levin, S. (1987). Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology*, 128(1), 11–45.

Kern, S., Müller, S., Hansen, N., Büche, D., Ocenasek, J., & Koumoutsakos, P. (2004). Learning probability distributions in continuous evolutionary algorithms–a comparative review. *Natural Computing*, 3(1), 77–112.

Kordon, A., Kalos, A., Castillo, F., Jordaan, E., Smits, G., & Kotanchek, M. (2005). Competitive advantages of evolutionary computation for industrial applications. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1 (pp. 166–173).

Kordon, A. K. (2010). *Applying Computational Intelligence—How to Create Value*. Springer.

Koza, J. (1992a). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge MA: MIT Press.

Koza, J. R. (1992b). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press.

Koza, J. R. & Poli, R. (2003). A genetic programming tutorial. In E. Burke & G. Kendall (Eds.), *Introductory Tutorials in Optimization, Search and Decision Support*, volume 8 chapter 8. Springer.

Laumanns, M., Thiele, L., Zitzler, E., & Deb, K. (2002). Archiving with guaranteed convergence and diversity in multi-objective optimization. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)* (pp. 439–447).

Lessmann, S., Stahlbock, R., & Crone, S. (2005). Optimizing hyper-parameters of support vector machines by genetic algorithms. In *International Conference on Artificial Intelligence* (pp. 74–82).

López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., & Birattari, M. (2011). The irace package, iterated race for automatic algorithm configuration. *IRIDIA, Université Libre de Bruxelles, Belgium, Tech. Rep. TR/IRIDIA/2011-004*.

Luke, S. (2013). *The ECJ Owner's Manual—A user manual for the ECJ Evolutionary Computation Library*. Department of Computer Science, George Mason University.

Luke, S. & Panait, L. (2001). A survey and comparison of tree generation algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)* (pp. 81–88).

Mathworks (2011). *Global Optimization Toolbox Documentation*.

McCarthy, J. (1960). Recursive functions of symbolic expressions and their computation by machine, part i. *Communications of the ACM*, 3(4), 184–195.

McGeoch, C. C. (2012). *A Guide to Experimental Algorithmics*. New York, NY, USA: Cambridge University Press, 1st edition.

Michalewicz, Z. & Fogel, D. B. (2004). *How to solve it: Modern Heuristics*. Berlin, Heidelberg, New York: Springer.

Michalewicz, Z. & Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evol. Comput.*, 4(1), 1–32.

Mikut, R. & Reischl, M. (2011). Data mining tools. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(5), 431–443.

Nguyen, T., Yang, S., & Branke, J. (2012). Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computing*, 6, 1–24.

Pappa, G. L. & Freitas, A. A., Eds. (2010). *Automating the Design of Data Mining Algorithms: An Evolutionary Computation Approach*. Springer.

Poli, R., Langdon, W. B., & McPhee, N. F. (2008). *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`. (With contributions by J. R. Koza).

R Core Team (2005). *R: A language and environment for statistical computing*. Technical report, ISBN 3-900051-07-0. R Foundation for Statistical Computing. Vienna, Austria, 2013. url: http://www. R-project. org.

Rapid-I (2010). *Rapid Miner 5.0 User Manual*.

Rechenberg, I. (1971). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, Department of Process Engineering, Technical University of Berlin, Germany.

Reeves, C. R. & Rowe, J. E. (2002). *Genetic algorithms-principles and perspectives: a guide to GA theory*, volume 20. Springer.

Rothlauf, F. (2011). *Design of Modern Heuristics: Principles and Application*. Springer.

Rowe, J. E. (2012). Genetic algorithm theory. In T. Soule & J. H. Moore (Eds.), *GECCO (Companion)* (pp. 917–940).: ACM.

Rudolph, G. (1997). *Convergence Properties of Evolutionary Algorithms*. Hamburg, Germany: Verlag Dr. Kovač.

Schmidt, M. & Lipson, H. (2009). Distilling free-form natural laws from experimental data. *science*, 324(5923), 81–85.

Schmidt, M. D. & Lipson, H. (2008). Data-mining dynamical systems: Automated symbolic system identification for exploratory analysis. In *Proc. of the Biennial ASME Conf. on Engineering Systems Design and Analysis, Haifa, Israel*.

Schwefel, H.-P. (1965). Kybernetische Evolution als Strategie der Exprimentellen Forschung in der Strömungstechnik. Master's thesis, Technical University of Berlin, Germany.

Schwefel, H.-P. (1975). *Evolutionsstrategie und numerische Optimierung*. Dr.-Ing. Dissertation, Technische Universität Berlin, Fachbereich Verfahrenstechnik.

Schwefel, H.-P. (1995). *Evolution and Optimum Seeking*. Sixth-Generation Computer Technology. New York NY: Wiley.

Searson, D. P., Leahy, D. E., & Willis, M. J. (2010). 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)*.

Smit, S. & Eiben, A. E. (2011). Multi-problem parameter tuning using BONESA. In J. Hao, P. Legrand, P. Collet, N. Monmarché, E. Lutton, & M. Schoenauer (Eds.), *Artificial Evolution, 10th International Conference Evolution Artificielle*, number 7401 in LNCS (pp. 222–233).: Springer.

Smits, G. F. & Kotanchek, M. (2005). Pareto-front exploitation in symbolic regression. In *Genetic Programming Theory and Practice II* (pp. 283–299). Springer.

Sywerda, G. (1989). Uniform crossover in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms* (pp. 2–9). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Tan, K. C., Yu, Q., & Lee, T. H. (2005). A distributed evolutionary classifier for knowledge discovery in data mining. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 35(2), 131–142.

Vladislavleva, E. (2008). *Model–based Problem Solving through Symbolic Regression via Pareto Genetic Programming*. PhD thesis, Tilburg University.

Vladislavleva, E., Friedrich, T., Neumann, F., & Wagner, M. (2013). Predicting the energy output of wind farms based on weather data: Important variables and their correlation. *Renewable Energy*, 50(0), 236 – 243.

Wagner, T. (2010). 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)* (pp.1̃).

Wegener, I. (2005). *Complexity Theory: Exploring the Limits of Efficient Algorithms*. Springer.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and Computing*, 4(2), 65–85.

Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., & Schmidhuber, J. (2011). *Natural Evolution Strategies*. Technical Report arxiv:1106.4487v1, arxiv.org.

Wierstra, D., Schaul, T., Peters, J., & Schmidhuber, J. (2008). Natural evolution strategies. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on* (pp. 3381–3387).: IEEE.

Witten, I. H. & Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

Yao, X., Liu, Y., & Lin, G. (1999). Evolutionary programming made faster. *Evolutionary Computation, IEEE Transactions on*, 3(2), 82–102.

Yu, T., Riolo, R., & Worzel, B. (2006). *Genetic programming: Theory and practice*. Springer.

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.