Evolutionary Algorithms

14

David Corne and Michael A. Lones

Contents

D. Corne (✉) · M. A. Lones
Heriot-Watt University, Edinburgh, UK
e-mail: d.w.corne@hw.ac.uk; mal1@hw.ac.uk

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_27

Abstract

Evolutionary algorithms (EAs) are population-based metaheuristics, originally inspired by aspects of natural evolution. Modern varieties incorporate a broad mixture of search mechanisms, and tend to blend inspiration from nature with pragmatic engineering concerns; however, all EAs essentially operate by maintaining a population of potential solutions and in some way artificially ‘evolving’ that population over time. Particularly well-known categories of EAs include genetic algorithms (GAs), Genetic Programming (GP), and Evolution Strategies (ES). EAs have proven very successful in practical applications, particularly those requiring solutions to combinatorial problems. EAs are highly flexible and can be configured to address any optimization task, without the requirements for reformulation and/or simplification that would be needed for other techniques. However, this flexibility goes hand in hand with a cost: the tailoring of an EA’s configuration and parameters, so as to provide robust performance for a given class of tasks, is often a complex and time-consuming process. This tailoring process is one of the many ongoing research areas associated with EAs.

Keywords

Population-based · Genetic algorithms · Evolutionary programming · Evolution strategies

Introduction

Evolutionary algorithms (EAs) are population-based metaheuristics. Historically, the design of EAs was motivated by observations about natural evolution in biological populations. Recent varieties of EA tend to include a broad mixture of influences in their design, although biological terminology is still in common use. The term “EA” is also sometimes extended to algorithms that are motivated by population-based aspects of EAs but which are not directly descended from traditional EAs, such as scatter search. The term evolutionary computation is also used to refer to EAs but usually as a generic term that includes optimization algorithms motivated by other processes but which generally involve a population of potential solutions adapting and improving over time. This includes algorithms inspired by other natural processes, such as ant colony optimization (ACO – inspired by the collective problem-solving behavior of social insects) and artificial immune systems (AIS – inspired by the adaptive molecular processes involved in the human immune system’s ability to recognize and destroy harmful agents). It also includes algorithms inspired by social behavior, such as particle swarm optimization (PSO); however, in recent years, the convention has shifted toward using the term “swarm intelligence” to describe PSO, ACO, and other algorithms inspired more by interaction than evolution. Although these algorithms often resemble EAs, this is not always the case, and they will not generally be discussed in this chapter. For a discussion of their commonalities and differences, the reader is referred to [1].

Over the years, EAs have become an extremely rich and diverse field of study. In part this arises from their inherent design flexibility – there are innumerable ways to specify an algorithm that operates according to the core concepts of evolution. In part this also arises from the fact that their performance tends to be highly problem dependent – an EA that works well on one task may well perform poorly on another, even quite similar, task. Arising from these factors and others, the sheer number of publications in this area is challenging for people new to the field. To address this, this chapter aims to give a concise overview of EAs and their application, with an emphasis on contemporary rather than historical usage.

The main classes of EA in contemporary usage are (in order of popularity) genetic algorithms (GAs), evolution strategies (ESs), differential evolution (DE), and estimation of distribution algorithms (EDAs). Multi-objective evolutionary algorithms (MOEAs), which generalize EAs to the multiple objective case, and memetic algorithms (MAs), which hybridize EAs with local search, are also popular, particularly within applied work. Special-purpose EAs, such as genetic programming (GP) and learning classifier systems (LCS), are also widely used. These are all discussed in this chapter.

Although these algorithms differ from each other in a number of respects, they are all based around the same core process. Each of them maintains a population of search points (known variously as candidate solutions, individuals, chromosomes, or agents). These are typically generated at random and are then iteratively evolved over a series of generations by applying variation operators and selection. Variation operators generate changes to members of the population, i.e., they carry out moves through the search space. After each generation, the objective value (or *fitness*) of each search point is calculated. Selection then removes the search points with the lowest objective values, meaning that only the best search points are maintained, and new search points are always derived from these. It is this combination of maintaining a population of search points and carrying out selection between search points that differentiates EAs from most other metaheuristics.

Each EA uses its own distinctive set of variation operators, which are sometimes inspired by the mutative and recombinative processes that generate diversity in biological evolution. The mutation operator resembles the generation of "moves" in other optimization algorithms and involves sampling the neighborhood around an existing search point in some fashion. A typical approach would be to randomly change one component of a solution, though a particular EA may use more than one kind of mutation operator. The recombination (or *crossover*) operator explores the region between two or more search points, for example, by randomly reassembling the components that make up two existing solutions. This process of searching the region between existing search points is also a distinctive feature of EAs, though its practical utility depends upon the structure of the search space. Some EAs, particularly "evolutionary programming" and older varieties of evolution strategy, do not use recombination at all.

Principal Algorithms

All EAs share certain common features, including in particular the broad concepts of *variation* and *selection*, as introduced in section "Introduction", operating over a series of *generations*. More fundamentally, all EAs work by seeking the "best" (in some sense) solutions they can find to a given optimization task. Put in another way, an EA is almost always used to find the solution data-structure x which optimizes a given function $f(x)$. For example, the task at hand may be straightforward numerical function optimization, where the candidate solution data-structure is a binary string that is interpreted as a list of real-valued parameters, and $f(x)$ is a mathematical function over those parameters whose result is a scalar value. Alternatively, the task at hand may be to find an ideal schedule for a collection of manufacturing tasks at a factory; in this case, the candidate solution data-structure might be an ordered sequence of task identifiers, and $f(x)$would be a program that simulates the schedule from the sequence, returning a vector of quality indicators including cost, time, and risk. Whatever the nature of the underlying data-structure being "evolved" and whatever the nature of $f(x)$, we can express a "canonical" EA in pseudocode as follows:

Preliminaries: determine a suitable way to represent solutions as data-structures, prepare the fitness function, and set $g = 0$.

```
Step 1: set $g = 0$, and generate and evaluate an initial population $S$g of candidate
    solutions;
Step 2: apply *selection* operators to produce a set of "parent" solutions $P$;
Step 3: apply variation operators to $P$ to produce a set of "chil" solutions $C$, and
    evaluate the fitness of each one;
Step 4: apply population update operations to the union of $Sg$ and C to produce
    $S_{g+}1$, and increment $g$
Step 5: if a termination criterion has not been reached, go to Step 2; otherwise,
    finish and return the fittest member of the population.
```

In the above pseudocode, steps 2–5 constitute a "generation," whose major components are selection, variation, and then the "population update" step that leads to renewal of the population, now ready to enter the next generation. Individual EAs vary greatly in each of these steps, including the "preliminaries" step. In the remainder of this section, we provide brief introductions to the principal classes of EA that are in current use and then discuss existing understanding of their performance and applicability.

Genetic Algorithms

Genetic algorithms, or GAs, are one of the earliest forms of EAs and remain widely used. Candidate solutions, often referred to as *chromosomes* in the GA literature, comprise a vector of decision variables. Nowadays, these variables tend to have a direct mapping to an optimization domain, with each decision variable (or *gene*) in the GA chromosome representing a value (or *allele*) that is to be optimized. However, it should be noted that historically GAs worked with binary strings, with real values encoded by multiple binary symbols, and that this practice is still sometimes used. GA solution vectors are either fixed-length or variable-length, with the former the more common of the two.

Given their long history, genetic algorithm implementations vary considerably. However, it is fairly common to use a mutation operator that changes each decision variable with a certain probability (values of 4–8% are typical, depending upon the problem domain). When the solution vector is a binary string, the effect of the mutation operator is simply to flip the value. More generally, if the solution vector is a "$k-$ary" string, in which each position can take any of a discrete set of k possible values, then the mutation operator is usually designed to choose a random new value from the available alphabet. If the solution vector is a string of real-valued parameters within a set range, the new value may be sampled from a uniform distribution in that range, or it may be sampled from a nonuniform (e.g., Gaussian) probability distribution centered around the current value. The latter is generally the preferred approach, since it leads to less disruptive change on average. Recombination is typically implemented using two-point or uniform crossover. Recombination tends to be applied at a high rate (e.g., 0.7, typically called the *crossover rate*); this means, for example, that when a variation operator is to be applied, the chance of this operator being a crossover operator will be 0.7; otherwise (depending on the algorithm), the operator may be the application of mutation to a single parent or simply copying a single parent. Two-point crossover chooses two *parent* solutions and two *crossover points* within the solutions. The values of the decision variables lying between these two points are then swapped to form two *child* solutions. Historically, "one-point" crossover was popular, which produced the first (second) child by copying the first (second) parent up to a randomly chosen point and then copied the second (first) parent thereafter. However, though more convenient for theoretical analyses, one-point crossover is rarely used in practice these days. Meanwhile, in uniform crossover, crossover points are created at each decision variable with a given probability. Other forms of crossover have also been used in GAs. Examples include line crossover and multi-parent crossover. Other variation operators, such as inversion, have been found useful for some problems.

Various forms of selection are used with GAs. Rank-based or tournament selection are generally preferred, since they maintain exploration better than the more traditional fitness-proportionate selection (e.g., roulette-wheel selection). Note, however, that the latter is still widely used. Rank-based selection involves ranking the population in terms of objective value. Population members are then chosen to become parents with a probability proportional to their rank. In tournament selection, a small group of solutions (typically three or four) are uniformly sampled from the population, and those with the highest objective value(s) become the parent(s) of the next child solution that is created. Tournament selection allows selective pressure to be easily varied by adjusting the tournament size.

Evolution Strategies and Evolutionary Programming

Evolutionary programming(EP) and evolution strategies (ES) also have a long history, starting earlier than the development of GAs. First described in the 1960s and 1970s, respectively (see [2] for a comprehensive account of the historical development), early ES and EP used only single-parent operators in the variation step (i.e., mutation). Meanwhile, EP was singular in focusing on using finite-state machines as the evolving data-structures, while ES soon introduced a recombination operator and championed the exploration of so-called "$m + n$" schemes, whereby a population of m parents would generate n children and the best n of the combined parents and children becomes the next generation of parents. In current research and practice, modern formulations of EP focus on numerical optimization over real-valued parameters and still eschew multi-parent operators in favor of paying attention to careful design of the mutation operator(s). Of the two styles, however, ES is more widely researched and deployed in practice, and particular variants of ES have shown particular prowess in numerical function optimization. Modern ESs incorporate strategies that carefully guide how the mutation operator is applied to each decision variable. Unlike GAs, ESs mutate every decision variable at each application of the operator and do so according to a set of *strategy parameters* that determine the magnitude of these changes. Strategy parameters usually control characteristics of probability distributions from which the new values of decision variables are drawn.

It is standard practice to adapt strategy parameters over the course of an ES run, the basic idea being that different types of move will be beneficial at different stages of search. Various techniques have been used to achieve this adaptation. Some of these involve applying a simple formula, e.g., the 1/5th rule, which involves increasing or decreasing the magnitude of changes based on the number of successful mutations that have recently been observed. Others are based around the idea of self-adaptation, which involves encoding the strategy parameters as additional decision variables and hence allowing evolution to come up with appropriate values. However, the most widely used contemporary approach is covariance matrix adaptation (CMA-ES), which uses a mechanism for estimating the directions of productive gradients within the search space and then applying moves in those directions. In this respect, CMA-ES has similarities with gradient-based optimization methods.

ESs use different recombination operators to GAs and often use more than two parents to create each child solution. For example, intermediate recombination gives a child solution the average values of each decision variable in each of the parent solutions. Weighted multi-recombination is similar but uses a weighted average, based on the fitness of each parent. Also unlike GAs, ESs tend to use deterministic rather than probabilistic selection mechanisms, whereby the best solutions in the population are always used as parents of the next generation.

Estimation of Distribution Algorithms

Like ESs, estimation of distribution algorithms [3], or EDAs, make use of probability distributions. However, rather than using them to describe a distribution of next moves, as an ES does, EDAs use them to describe a distribution of next sample solutions. The basic mechanism is quite simple. As for most EAs, the initial population of solutions is (typically) sampled from a uniform distribution. Selection is then used to remove the poorer members of this population, and a probability distribution is then constructed that attempts to model the statistics of the relatively high-fitness sample solutions that remain in the population. Importantly, this distribution is constructed in such a way that it "generalizes" the population members suitably well. The next generation of solutions is then constructed by sampling from this distribution. So, if the distribution was highly peaked around the existing samples, for example, the next generation would explore very little beyond the previous one. This pattern of building a distribution, sampling, and selection is then iterated in the usual generational fashion, with the hope that the final probability distribution will characterize solutions that are, or are close to, globally optimal.

While an EDA may be used with any kind of probability distribution, in practice, it is necessary to choose a distribution that induces an appropriate trade-off between efficiency and expressiveness. More expressive models, such as Bayesian networks and Markov models, can capture dependencies between decision variables but can be expensive to construct and sample from. Simple univariate distributions, by comparison, are cheap to build and sample from but are unable to capture dependencies between variables. This trade-off is reflected in the range of EDAs in common use. Population-based incremental learning (PBIL) and the compact genetic algorithm (CGA) are both examples of computationally efficient EDAs that build simple univariate models based on discrete variables. Because of their simplicity, they can be applied to large problem instances. Bayesian optimization algorithms (BOAs) lie at the other end of the spectrum: these can express dependencies between variables, and certain varieties can be applied to both discrete and continuous variables, but they are far more demanding of computational resources.

Differential Evolution

Differential evolution [4,5], or DE, is a relatively recent EA formulation which uses a mechanism for adaptive search that does not make use of probability distributions. While its basic mechanism is similar to a GA, its mutation operator is quite different, using a geometric approach that is motivated by the moves performed in the Nelder-Mead simplex search method. This involves selecting two existing search points from the population, taking their vector difference, scaling this by a constant F, and then adding this to a third search point, again sampled randomly from the population. Following mutation, DE's crossover operator recombines the mutated search point (the *mutant vector*) with another existing search point (the *target vector*), replacing it if the child solution (known as a *trial vector*) is of equal or greater objective value. There are two standard forms of crossover [6]: exponential crossover and binomial crossover, which closely resemble GA two-point crossover and uniform crossover, respectively. The comparisons between target vector and trial vector play the same role as the selection mechanism in a GA or ES. Since DE requires each existing solution to be used once as a target vector, the whole population is replaced in the course of applying crossover.

An advantage of using simplex-like mutations in DE is that the algorithm is largely self-adapting, with moves automatically becoming smaller in each dimension as the population converges. More generally, the authors of the method have claimed that this sort of self-adaptation means that the size and direction of moves are automatically matched to the search landscape, a phenomenon they term *contour matching*. When compared to CMA-ES, for example, this means that the algorithm has few parameters and is relatively easy to implement.

Performance Comparisons

Fair comparisons of optimization algorithms are inherently challenging [7] and arguably unachievable. Nevertheless, there have been some attempts to understand the comparative performance of different EAs, particularly within the domain of continuous optimization. In particular, a series of workshops held at two of the largest annual EA conferences, CEC and GECCO, have sought to define benchmark suites of real-valued function optimization problems suitable for comparing EAs (and other optimizers) [8–10]. Using these benchmarks, a number of authors have shown their algorithms to perform better than others, including variants of CMA-ES and DE (see [4]). It should be borne in mind that these are not exhaustive studies, either in terms of problems or approaches. The "no free lunch theorem"(NFLT) [11] may also be considered when attempting to generalize these results to a wider spectrum of problems, although, in itself, the NFLT does not apply in the case of comparisons based on the standard suites of test problems (since those suites are not closed under permutation [12]). A nice example of the perils of comparison study in this field is shown by a recent study that showed how quite different conclusions could be drawn from a comparative study by changing minimization problems into maximization problems [13].

Common Variants

The general purpose EAs introduced in the last section are applicable to a wide range of problems. However, over the course of EA history, algorithmic variants have been developed to deal with the characteristics of particular categories of problem. Some of these categories are quite broad, for example, problems with multiple solutions.

Others are more specific, such as discrete optimization problems. In this section, we discuss a number of these EA variants, focusing on those which are commonly used to solve real-world optimization problems.

Alternative Representations

In common with other optimization algorithms, most EAs are designed to work with and optimize vectors (or, equivalently, lists or arrays) of decision variables. Solutions to many kinds of problems can be represented, either directly or indirectly, in this form. However, EAs are not limited to working with vectors, and there are often advantages to working directly with representations that are more natural for the problem domain: for example, matrices [14], trees [15], graphs [16], rule sets, etc. The general approach is the same as for the EAs discussed in the previous section, except that specialized initialization routines and variation operators are used to randomly create, mutate, and recombine instances of the appropriate solution representation. These variants are typically based around GAs or ESs, since these two classes of EA can be readily adapted to use alternative solution representations. Nevertheless, DE [17] and EDA [18] have also been used successfully with other representations.

Genetic programming (GP) [15] is a well-known GA variant in which each candidate solution is a *program*; in early GP this was invariably achieved by encoding programs as tree-structures, in which each "node" in the tree corresponds to a function whose input parameters are the results returned by its child-nodes and which itself returns a result to its parent nodes (if any). GP is still mostly used to optimize computer programs or mathematical expressions, often expressed as tree-structures. However alternative ways to encode programs are now often explored in GP, such as so-called linear GP, in which programs are represented as a sequence of parameterized instructions interacting via registers. A particularly common current use of GP is *symbolic regression*, which involves finding a mathematical expression that fits a particular data set. Unlike standard mathematical approaches to regression, such as curve fitting, GP makes relatively few assumptions about the function that generated the data, allowing a wide exploration of the space of possible solutions. GP is also widely used for solving classification problems. More generally, the GP community is interested in automatic programming, i.e., finding computer programs that solve a particular task, and there are many variants of GP that use particular forms of program representation. See [19] and [20] for overviews.

Some EAs work with two different solution representations, using one of these when creating and manipulating search points, the other when evaluating search points, and a mapping process that converts the former into the latter [21]. Many of these approaches are motivated by biology, and hence this process is known as a *genotype-phenotype* mapping, with the representation used during search termed the *genotype* and the representation used for evaluation termed the *phenotype*. This approach can be used when the natural representation for a domain is not well suited to being evolved, i.e., where mutation and recombination do not lead to productive solutions. This approach has also been widely used for generating complex structures, such as large neural networks [22], where a genotype representation can be chosen that compresses repetitive features such as symmetry and modularity. This is arguably an area in which EAs benefit from their relationship to biology, since biology provides a ready source of information on how to represent complex structures in an evolvable way.

Hybridization with Local Search

EAs are often considered to be global search algorithms, since they explore a relatively wide region of the search space and are relatively good at escaping local optima. However, their convergence to optimal solutions can be relatively slow when compared to local search algorithms. For this reason, EAs are often hybridized with local search, using it to locally optimize members of the population at regular intervals, hence speeding up convergence. Although the resulting hybrid algorithms are known by various names, the term *memetic algorithm* [23] (MA) has become popular in recent years. Memetic, in this case, refers to an analogy between the role of local search in these algorithms and the role of within-generation learning in biological systems, though the majority of memetic algorithms have no particular biological justification beyond this. In principle, these algorithms may involve hybridizing an EA with any or with multiple local search algorithms and consequently are very diverse. For a recent review, see [24].

Beyond hastening convergence, MAs are also seen as a means of introducing domain knowledge into EAs. This is done through the use of specialized local search operators that are relevant to a particular domain. For example, this approach underlies the success of MAs in the area of discrete optimization [25]. Related to the idea of problem specialization in memetic algorithms is the concept of hyper-heuristics in EAs, which has developed some traction in recent years [26]. Generally speaking, hyper-heuristics are applicable to domains in which a variety of so-called low-level heuristics exist (or can be invented) to build quick, good solutions. In the job shop scheduling problem, for example, "shortest-process-time" and "earliest-available-machine" are two examples of low-level "dispatch" heuristics that, when iterated, can build a single solution quickly. Hyper-heuristics are essentially mechanisms used to explore *combinations* of such lower-level heuristic strategies. The term hyper-heuristics is also used to describe the cases in which an EA (often GP) both creates anew and combines such low-level heuristics. In a nutshell, the broad idea of hyper-heuristics is to search a space of algorithms that can solve a class of problems, rather than search the space of solutions directly for a single problem instance.

Multimodal Optimization

An advantage of maintaining a population of search points is that EAs can be readily applied to multimodal optimization problems in which there is more than one solution of interest. However, effective multimodal optimization generally requires some modification to the EA's underlying behavior, since, although EAs explore diverse areas of the search space, they eventually converge to fairly small areas. This behavior can be mitigated, to an extent, by varying the global selection pressure used when choosing parents; for example, in the case of tournament selection, the tournament size can be made small, increasing the likelihood that less fit members of the population will contribute to the next generation of search points. While this increases exploration, it decreases exploitation: meaning that multiple solutions may be found, but they are less likely to be optimal. Since EAs are stochastic, and there is the potential for them to converge on different optima during different runs, another simple approach to finding multiple solutions is to run an EA multiple times. However, there is no guarantee that all optima will be explored, and algorithmic biases (such as the manner in which the initial population is generated) may favor some solutions over others.

A more effective approach is to use some kind of *niching* technique [27]. These aim to preserve global diversity in the population, but without lowering local selective pressure. Niching approaches are motivated by the biological concept of evolutionary niches, in which species compete within a niche but not between niches. In optimization terms, a niche is a local region within the search space that contains a solution of interest, and the aim is for the population to be distributed across all the relevant niches. Niching has been studied for some time in GAs, and techniques include crowding [28], fitness sharing [29], spatial segregation [30], and clustering [14]. For comparative studies, see [6] and [8]. A simple but effective example of niching is probabilistic crowding [28]. This works at the operator level and always replaces parent solutions with their children, meaning that search points are usually replaced with nearby search points and the population remains spread across the search space. Similar techniques have also been developed for use in DE. Niching is less commonly used in ESs, in part due to their use of smaller populations, though examples do exist [31]. It is also common to use multi-objective evolutionary algorithms (see below) to solve multimodal problems, since these algorithms often have effective mechanisms for preserving population diversity.

Multi-objective Optimization

Multi-objective EAs, or MOEAs, are used to solve problems which have multiple and often conflicting objectives. A central concept for MOEAs, and multi-objective optimization in general, is that of a non-dominated solution. This is a solution which is no worse than any of the other solutions within the population when all objectives are taken into account, and the aim of an MOEA is to build and maintain a population of non-dominated solutions that cover all trade-offs between the objectives. This is known as the Pareto optimal front. Exactly how this is achieved varies between MOEAs. However, a well-known example is NSGA-II (non-dominating sorting genetic algorithm) [32]. Prior to selection, NSGA-II ranks all solutions in terms of dominance: those which are non-dominated are assigned rank 1, those which are only dominated by rank 1 solutions are assigned rank 2, etc. The population is then ordered by rank, and by a measure of crowding distance within ranks, and the first half of the ordered population is copied directly into the next generation. The remainder of the population is then filled by breeding, with parents selected from the higher ranks. Hence, non-dominated solutions are preserved between generations, and new solutions are explored via interbreeding, resulting in a diverse set of non-dominated solutions that approximate the Pareto optimal front.

The core challenge faced by multi-objective optimization (and absent from single-objective optimization) is how to rank candidate solutions in a way that leads to effective selection pressure, especially when the entire population (or most of it) may be non-dominated. Another way in which multi-objective optimization differs from single-objective optimization is in the nature of the "best-so-far solution." In single-objective optimization, the "best-so-far" solution is trivial to define and to keep track of; in multi-objective optimization, the situation is vastly different: the solution is, technically, the entire Pareto front, which is usually a set of solutions, whose cardinality may vary from one to the entire search space. For MOEAs, this leads to certain technical issues which are invariably addressed by maintaining an *archive* of non-dominated solutions; this archive simply keeps track of the "best-so-far" approximation to the Pareto front but is also often used as a reservoir for selection of parents. Approaches to the main challenge – how to apply effective selection pressure among the current population – are far more varied. While the approach taken by NSGA-II, as detailed above, is a common and quite successful one, many other styles of MOEA exist, which take different approaches to this central question. In PAES [33], for example, there is only a single "current" population member. Selection is consequently simplified; however, the challenge shifts to the question of whether or not to update the current solution with a newly generated one when the two are non-dominated; PAES makes this decision with the aid of its archive, preferring to explore new areas of the search space than to stay close to solutions already in the archive. Meanwhile, a different breed of MOEAs in this respect is represented by MOEA/D [34]; bypassing the need to distinguish between non-dominated solutions for selection purposes, MOEA/D "decomposes" a multi-objective problem into many single-objective simplifications of it, each involving a different weighting of the objectives. MOEA/D conducts these single-objective searches in parallel (typically using a local search mechanism) and organizes occasional communication between them, as well as bookkeeping activities that build and maintain the archive. Effectively, each of MOEA/D's single-objective searches explores a different area of the Pareto front. There are many other approaches, and MOEAs are becoming increasingly used as it becomes recognized that real-world problems are almost invariably multi-objective in nature. Further discussion of the latter point, as well as a first introduction to MOEAs, may be found in [35], while an example of a fairly recent review of MOEAs may be found in [36].

Dynamic Optimization

So far, our discussion of optimization has only considered problems in which the search space remains fixed. In many real-world problems, this is not the case, and various EA approaches are used to handle these situations. Dynamic optimization is an area in which EAs might be expected to perform relatively well, since the natural diversity present in their populations provides a recovery mechanism that can respond to slow changes in the optimization landscape. This is especially the case when diversity maintenance techniques are implemented, such as those already discussed in the sections on multimodal and multi-objective optimization. However, this diversity may be insufficient when the optimization targets change rapidly or abruptly. A simple solution in this situation is to inject extra diversity into the population when a change is detected, for instance, by adapting the variation operators so that larger moves are made. Detection of change can be done by reevaluating a proportion of the population, looking for significant changes in fitness.

A variety of more elaborate approaches have been developed to handle dynamic optimization in EAs. An approach inspired by biological systems is to use redundancy in the encoding of a solution. Rather than replacing components of a solution when variation operators are applied, this allows old components to become recessive, i.e., to remain present within the solution but not be expressed during evaluation. Later in the evolutionary process, these components can become reactivated, in effect providing a mechanism to backtrack to previous search points. This is particularly useful when changes in the search space are cyclic. A well-known example is the use of *multiploidy* in GAs [37], where each solution has multiple chromosomes (only one of which is dominant) and variation operators are able to move information between chromosomes. Other approaches to handling dynamic search spaces include predicting change and using multiple populations; see [38] for a recent review.

Coevolving Solutions

Coevolutionary algorithms [39] are motivated by the interactions that occur between species during the course of biological evolution and the roles these interactions are thought to play in the evolution of complex organisms. Most coevolutionary algorithms use multiple populations, one per *species*. Coevolutionary relationships in biology can be cooperative or competitive. The latter class are particularly well known and are encapsulated in the idea of predator-prey patterns of evolution, where an arms race between two species can lead to the rapid emergence of complex adaptations. Similar ideas have been explored in EAs, the classic example being the coevolution of sorting networks and sorting algorithms [40]: the discovery of harder problems (the sorting networks in the first population) leads to selective pressure to discover better solutions (the sorting algorithms in the second population), which leads to selective pressure to discover harder problems and so on. Competitive coevolution can be used to solve hard problems and is also useful in circumstances where a fitness function cannot be defined. However, competitive coevolution is known to be difficult to control, and pathological situations can lead to ineffective search. See [39] for a review.

Cooperative coevolution, by comparison, is seen as a useful mechanism for breaking down large problems into more tractable chunks [41]. The idea is that a solution to a problem is divided into sub-components. Each of these sub-components is then evolved in a separate population, with its objective value dependent upon how compatible it is with sub-components being evolved in other populations. Following this, the coadapted sub-components are then assembled to form a complete solution. In [42], for example, the authors describe how a cooperative coevolutionary variant of DE can be used to solve numerical optimization problems with up to 1000 variables. Cooperative coevolution can also take place within a single population. An example of this is a Michigan-style learning classifier system (LCS), a form of EA that coevolves a population of rules that can collectively solve difficult problems in classification and machine learning [43].

Applying Evolutionary Algorithms

Choosing a Methodology

It can be difficult to choose which EA to use for a particular task, since there are many different EAs in common use and relatively little in the way of objective comparative guidance. In practice, it may be necessary to try out different EAs to find out which is the best match to a problem, especially when the problem is poorly understood. However, given whatever is known about the problem at hand, it might be possible to leverage existing understanding of the strengths and weaknesses of particular algorithm frameworks. Some guidance on this matter is available in studies of comparative performance mentioned at the end of section "Introduction". It is hoped that section "Principal Algorithms" also provides useful pointers if the problem is multimodal, multi-objective, dynamic, or unusually large and complex. It is also notable that multi-objective and memetic algorithms, in particular, have become popular for solving difficult real-world problems.

Choosing Parameters

EAs invariably have many parameters, and once an algorithm has been selected, it is normal to carry out parameter tuning in order to obtain a better fit between the algorithm and the problem. It can be challenging to obtain optimal parameter settings, since parameters are typically both numerous and not independent of one another. DE, for instance, is notable for having relatively few parameters, and this is often portrayed as a strength of the method. However, EAs are relatively forgiving, and good performance is likely to be possible with nonoptimal parameter settings. Nevertheless, guidance is available for choosing the settings of certain parameters [44], and a number of techniques have been developed for automating the choice of parameter settings [45].

Software Tools

Tools support is an important issue for many practitioners, and a particular EA methodology is likely to be more appealing if it has a mature supported implementation. Tools support is also important if it is necessary to handcraft a new algorithm to solve a particular problem, and in this situation the language used by the tool may also be a significant concern. Table 1 summarizes the features of some of the better known EA tools. GAs are widely supported by all of these. ES support is also widely available, though EvA2 stands out in this regard, with implementations of a wide range of ES variants. DE and EDAs are more recent algorithms, and this is reflected by fewer mature tools. However, EvA2 is again notable for having an implementation of BOA and other EDAs. GP support is offered by a number of these tools, with ECJ implementing a particularly wide range of GP variants. Most also offer support for multimodal and multi-objective approaches, though MOEA Framework stands out for the latter. All of these tools allow custom code to be written. Most use Java or C + +, though DEAP is notable as a mature Python implementation and HeuristicLab is available for C# users.

Table 1 Open-source EA frameworks

Tool	Language	Summary
DEAP https://code.google.com/p/deap/	Python	Distributed Evolutionary Algorithms in Python offers good support for GAs and ESs. Also implements GP and MOEAs
ECJ http://cs.gmu.edu/~eclab/projects/ecj/	Java	Continuously developed since 1998, Evolutionary Computation in Java has particular strength in GP but also implements GAs, DE, and MOEAs
EO http://eodev.sourceforge.net	C++	Evolving Objects is an established general-purpose EA library with implementations of GAs, GP, ESs, and EDAs
EvA2 http://www.ra.cs.uni-tuebingen.de/software/JavaEvA/	Java	EvA2 is a general-purpose EA framework but has particular strengths in ESs and EDAs, including BOA
HeuristicLab http://dev.heuristiclab.com/	C#	HeuristicLab implements many of the common EA varieties and also has support for other population-based and local search metaheuristics
MOEA framework http://www.moeaframework.org/	Java	A relatively new EA framework with considerable strength in MOEAs and multi-objective variants of DE and GP
OpenBEAGLE https://code.google.com/p/beagle/	C++	A long established EA framework with good support for GAs and ESs. Also implements GP and NSGA-II

Case Studies

Evolutionary Algorithms at Large

Now more than half a century since the first appearance of "EA"-style algorithms in the research literature (widely considered to be [46]), EAs have penetrated almost every area of science and industry and are regularly used in solving an immense range of optimization problems.

In terms of broad classes of problem, EAs have enabled practitioners and researchers to make particular headway in *combinatorial optimization* which, unlike *numerical optimization*, for example, was hitherto poorly served by classical algorithms. In combinatorial optimization, the task is typically to find an ideal permutation (or otherwise constrained arrangement) of a set of entities – such as a sequence of customer visits for a delivery vehicle or a sequence of production tasks for each of a set of machines in a factory. Before EAs, the primary approach used to solve such problems was integer programming and constraint programming (and variations thereon),which tend to require a complex (and often tortuous) problem reformulation step. EAs, however, provide a far more accessible and flexible approach to addressing such problems and are now commonly used in practice for combinatorial tasks such as vehicle routing, job shop scheduling, and facility allocation [47].

Meanwhile, beyond combinatorial optimization, the inherent flexibility of EAs has led to their use, to at least some extent, in every conceivable area of science, enterprise, and industry in which one or more important tasks can be formulated in terms of optimization and/or design. To name just a few of the areas in which EAs have had much impact, we can list aeronautical and automotive design [48], bioinformatics and biotechnology [49], chemical engineering [50], creative pursuits [51], finance and investment [52], manufacturing [53], and structural design [54]. To select a small number of case studies could not serve to characterize the true diversity of applied EAs. We therefore duck that challenge and take the liberty of concluding this chapter by providing two case studies from the authors' recent work, illustrating how some EA approaches are being applied to diverse and challenging optimization problems in just one corner of science.

Using Niching and Coevolution to Understand Gene Regulation

Understanding gene expression is fundamental to understanding living processes. The expression of each gene in an organism is determined by the binding of special proteins, called transcription factors, within a region of DNA upstream of its coding region. In higher organisms, such as humans, this regulatory region typically contains around 5–10 transcription factor binding sites. Characterizing these bindings sites, both individually and in combination, is a fundamental part of reconstructing (and ultimately controlling) the genetic networks that underlie biological function.

Identifying binding sites is often reduced to an optimization problem that involves constructing a matrix model of the occurrence of each DNA base at each position within a short region of DNA. Candidate solutions to this problem can be evaluated by scanning them along the regulatory regions of groups of genes which are known to be expressed at a certain time or within a certain cell type, looking for matches to patterns embedded in the sequences. In most cases, this is a multimodal problem, since multiple binding sites are likely to be relevant to a particular regulatory context, and it is important to be able to identify these different optima. However, these binding sites can vary quite considerably in their degree of conservation, meaning that the objective values of different optima also vary considerably.

Identifying and preserving different modes within a multimodal search space is a challenging problem, especially when they have different relative finesses. In section "Common Variants", we discussed the idea of niching within the populations of EAs as a means of addressing multimodal problems. In [14] we used an EA furnished with a particular form of niching, termed population clustering, that uses a clustering algorithm to identify and preserve the different modes present within an evolving population of solutions. This allowed a number of different binding sites to be characterized and preserved during a single run. Compared to other forms of niching, it also had the benefit of explicitly identifying these different groups of solutions, allowing the progress of search to be visualized and for clustering parameters to be dynamically modified by an expert user. Even in the absence of dynamic modification, however, this was effective at identifying the clusters of binding sites within comparatively long regions of DNA.

Identifying binding sites is only one part of the problem. Another important aim is to understand the interactions between different binding sites during gene regulation. In [55], we used coevolution to explore solutions to this problem. This involved coevolving two populations, the first containing matrix models of binding sites and the second containing Boolean expressions describing their co-occurrence within binding regions. Members of the binding site population were used as leaves within the Boolean expressions. In essence, the problem of identifying binding sites and their co-expression was decomposed into two problems which were then solved in parallel, using coevolution to provide feedback between the populations. In comparison to a more traditional approach, which would involve sequentially learning binding sites and then learning their interactions, this allows search to be directed toward solutions that interact well with other solutions during the course of search. Such solutions, in turn, are more likely to be meaningful. This approach proved effective for reverse engineering the regulatory rules underlying differential gene expression within tissues. More surprisingly, it also provided a mechanism for solving harder instances of the single binding site optimization problem, with coevolution provided a means of implicitly decomposing the matrices associated with these harder problems.

Evolving Classifiers for Parkinson's Disease Diagnosis

This second case study concerns a problem in which the optimal representation for solutions is not clear in advance, requiring experimentation with different kinds of solution representation. As discussed earlier, EAs are entirely flexible in this regard. The problem involves building diagnostic classifiers for Parkinson's disease. These are required to reach their decision based on time series movement data recorded while patients and age-matched control were undergoing clinical assessments of motor function. Motor aspects of Parkinson's are incompletely understood, making it unclear what kind of features of the data are important.

To address this, we used EAs to explore two different relatively unconstrained classifier models. Initially we considered a GP-based approach, using it to discover mathematical expressions that describe overrepresented patterns of movement embedded in short segments of the time series data, i.e., a form of symbolic regression [56]. An advantage of this approach is that the resulting expressions were relatively interpretable, allowing us to gain insight into the basis of classifications and then pass this information on to our clinical partners. In particular, analysis of the evolved expressions identified specific aspects of the closing phase of a "finger tap" movement as highly discriminatory of Parkinson's disease patients versus control. These factors alone were indeed more discriminatory than standard metrics; however, overall classification performance was not quite as good as that of trained clinicians.

We then considered a more unusual method of representing programs, artificial biochemical networks [57]. These are abstract executable models of the networks of biochemical interactions that underlie the function of biological cells. In a nutshell, they attempt to capture the representation which biological evolution has selected to optimize complex behaviors, with the hypothesis that this makes them particularly suitable for use with EAs. Meanwhile, the use of an EA to search through the space of possible artificial biochemical network classifiers represents, in itself, a major and commonly understood strength of EAs: they can be tailored and deployed effectively with relative ease despite the complexity and diversity of the structure they are being used to evolve. By using this approach, we were now able to find classifiers that produced comparable performance to that of trained clinicians [58]; with accuracy at around 90% overall, accuracy was comparable to the diagnostic accuracies found in clinical diagnosis, and significantly higher than those found in primary and non-expert secondary care.

Conclusion

In this chapter we have provided a broad introduction to and overview of evolutionary algorithms, and the many varieties of them that appear in modern research and practice. We have seen that there are a handful of key EA 'families', such as 'genetic algorithms' and 'differential evolution'; meanwhile, to some extent cutting across the principal types of EAs, there are several algorithm features and variants (such as co-evolution, or hybridisation with local search) that are often 'grafted' into an EA, in order to boost performance on a particular class of problems (or perhaps just for research purposes). Towards the end, we considered the real-world application of EAs, reporting that they are generally highly successful, and are growing to be common tools used for optimisation tasks in science and industry. However, for someone new to EAs, an early question might well be: "There are so many types and variants of EAs – which should I use?" In this chapter, we have effectively promised that EAs are likely to be a successful approach to any given optimization task – particularly if that task is not well served by other available schemes or heuristics. However, we have not necessarily made it easy for a novice to deliver on that promise. A common theme has been the very wide variety of EA designs that have been developed, and this great variety can be a barrier for those new to the field, who aim to understand the key aspects of EAs and how best to apply them to any given task. In closing, we attempt to provide some support for the novice via the following perspective: EAs should not be seen as a collection of algorithms; instead, they are far more fruitfully viewed as an approach to the engineering of problem-specific (or problem-class specific) optimization algorithms. The key lesson for practitioners, from half a century of research and practice in EAs, is, broadly speaking, that concepts from evolution (population, selection, variation) can be very powerful tools in optimization; but to achieve ideal performance on a given task, one must be pragmatic, and often creative, about the details.

Cross-References

- ▶ A History of Metaheuristics
- ▶ Evolution Strategies
- ▶ Evolutionary Algorithms for the Inverse Protein Folding Problem
- ▶ Genetic Algorithms
- ▶ Hyper-heuristics
- ▶ Memetic Algorithms
- ▶ Multi-objective Optimization
- ▶ Particle Swarm Methods

References

1. Lones MA (2014) Metaheuristics in nature-inspired algorithms. In: Proceedings of genetic and evolutionary computation conference (GECCO 2014), workshop on metaheuristic design patterns (MetaDeeP). ACM, pp 1419–1422
2. Fogel DB (1998) Evolutionary computation: the fossil record. Wiley-IEEE Press, Piscataway
3. Hauschild M, Pelikan M (2011) An introduction and survey of estimation of distribution algorithms. Swarm Evol Comput 1(3):111–128. https://doi.org/10.1016/j.swevo.2011.08.003. Available: http://www.sciencedirect.com/science/article/pii/S2210650211000435

4. Das S, Suganthan PN (2011) Differential evolution: a survey of the state-of-the-art. IEEE Trans Evol Comput 15(1):4–31. https://doi.org/10.1109/TEVC.2010.2059031. Available: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5601760&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs~all.jsp%3Farnumber%3D5601760
5. Storn R, Price K (1997) Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 11(4):341–359. https://doi.org/1008202821328. Available: http://link.springer.com/article/10.1023%2FA%3A1008202821328#page-1.
6. Zaharie D (2009) Influence of crossover on the behavior of differential evolution algorithms. Appl Soft Comput 9(3):1126–1138. https://doi.org/10.1016/j.asoc.2009.02.012. Available: http://www.sciencedirect.com/science/article/pii/S1568494609000325
7. García S, Molina D, Lozano M, Herrera F (2009) A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. J Heuristics 15(6):617–644. https://doi.org/10.1007/s10732-008-9080-4. Available: http://link.springer.com/article/10.1007/s10732-008-9080-4
8. Hansen N, Auger A, Finck S, Ros R (2010) Real-parameter black-box optimization benchmarking 2010: experimental setup. INRIA research report No. 7215. INRIA
9. Liang J, Qu B, Suganthan P, Hernández-Díaz A (2013) Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization. Technical report 201212. Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, pp 3–18
10. Tang K, Li X, Suganthan PN, Yang Z, Weise T (2009) Benchmark functions for the CEC'2010 special session and competition on large-scale global optimization. Technical report. Nature Inspired Computation and Applications Laboratory, University of Science and Technology of China
11. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82. https://doi.org/10.1109/4235.585893. Available: http://ieeexplore.ieee.org/xpls/abs~all.jsp?arnumber=585893
12. Igel C, Toussaint M (2003) On classes of functions for which no free lunch results hold. Inf Process Lett 86(6):317–321
13. Piotrowski AP (2015) Regarding the rankings of optimization heuristics based on artificially-constructed benchmark functions. Inf Sci 297:191–201. Available: http://www.sciencedirect.com/science/article/pii/S0020025514010937
14. Lones MA, Tyrrell AM (2007) Regulatory motif discovery using a population clustering evolutionary algorithm. IEEE/ACM Trans Comput Biol Bioinform 4(3):403–414. https://doi.org/10.1109/tcbb.2007.1044. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4288066
15. Koza J (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge
16. Miller JF (2011) Cartesian genetic programming. https://doi.org/10.1007/978-3-642-17310-3_2
17. Veenhuis CB (2009) Tree based differential evolution. Lect Notes Comput Sci 5481:208–219
18. Kim K, Shan Y, Nguyen X, McKay RI (2014) Probabilistic model building in genetic programming: a critical review. Genet Program Evolvable Mach 15(2):115–167. https://doi.org/10.1007/s10710-013-9205-x. Available: http://link.springer.com/article/10.1007/s10710-013-9205-x
19. Poli R, Langdon W, McPhee NF (2008) A field guide to genetic programming. Published via http://lulu.com
20. Luke S (2013) Essentials of metaheuristics. Published via http://lulu.com
21. Stanley KO, Miikkulainen R (2003) A taxonomy for artificial embryogeny. Artif Life 9(2):93–130. https://doi.org/10.1162/106454603322221487. Available: http://www.mitpressjournals.org/doi/abs/10.1162/106454603322221487 (pages 94 and 95)

22. Floreano D, Dürr P, Mattiussi C (2008) Neuroevolution: from architectures to learning. Evol Intell 1(1):47–62. https://doi.org/10.1007/s12065-007-0002-4. Available: http://link.springer.com/article/10.1007/s12065-007-0002-4
23. Moscato P (1989) On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Caltech concurrent computation program, C3P report 826
24. Neri F, Cotta C (2012) Memetic algorithms and memetic computing optimization: a literature review. Swarm Evol Comput 2:1–14. https://doi.org/10.1016/j.swevo.2011.11.003. Available: http://www.sciencedirect.com/science/article/pii/S2210650211000691
25. Hao J (2012) Memetic algorithms in discrete optimization. In: Neri F, Cotta C, Moscato P (eds) Handbook of memetic algorithms. Springer, Berlin/Heidelberg. https://doi.org/10.1007/978-3-642-23247-3_6
26. Ross P (2005) Hyper-heuristics. In: Search methodologies. Springer, Berlin, pp 529–556
27. Singh G, Deb K (2006) Comparison of multi-modal optimization algorithms based on evolutionary algorithms. ACM, New York. https://doi.org/10.1145/1143997.1144200
28. Mengshoel OJ, Goldberg DE (2008) The crowding approach to niching in genetic algorithms. Evol Comput 16(3):315–354. https://doi.org/10.1162/evco.2008.16.3.315. Available: http://www.mitpressjournals.org/doi/abs/10.1162/evco.2008.16.3.315
29. Sareni B, Krahenbuhl L (1998) Fitness sharing and niching methods revisited. IEEE Trans Evol Comput 2(3):97–106. https://doi.org/10.1109/4235.735432. Available: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=735432&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel4%2F4235%2F15834%2F00735432.pdf%3Farnumber%3D735432
30. Lim T (2014) Structured population genetic algorithms: a literature survey. Artif Intell Rev 41(3):385–399. https://doi.org/10.1007/s10462-012-9314-6. Available: http://link.springer.com/article/10.1007%2Fs10462-012-9314-6
31. Shir OM, Back T (2005) Dynamic niching in evolution strategies with covariance matrix adaptation. https://doi.org/10.1109/CEC.2005.1555018
32. Deb K, Pratap A, Agarwal S, Meyarivan TAMT (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197
33. Knowles J, Corne D (1999) The pareto archived evolution strategy: a new baseline algorithm for paretomultiobjective optimisation. In: Proceedings of the 1999 congress on evolutionary computation (CEC'99), vol 1. IEEE
34. Zhang Q, Li H (2007) MOEA/D: a multiobjective evolutionary algorithm based on decomposition. IEEE Trans Evol Comput 11(6):712–731
35. Corne DW, Deb K, Fleming PJ, Knowles JD (2003) The good of the many outweighs the good of the one: evolutionary multi-objective optimization. IEEE Connect Newslett 1(1):9–13
36. Zhou A, Qu B, Li H, Zhao S, Suganthan PN, Zhang Q (2011) Multiobjective evolutionary algorithms: a survey of the state of the art. Swarm Evol Comput 1(1):32–49. https://doi.org/10.1016/j.swevo.2011.03.001. Available: http://www.sciencedirect.com/science/article/pii/S2210650211000058
37. Goldberg D, Smith R (1987) Nonstationary function optimization using genetic algorithm with dominance and diploidy. In: Proceedings of the second international conference on genetic algorithms and their application (ICGA). Laurence Erlbaum Associates, pp 59–68
38. Nguyen TT, Yang S, Branke J (2012) Evolutionary dynamic optimization: a survey of the state of the art. Swarm Evol Comput 6:1–24. https://doi.org/10.1016/j.swevo.2012.05.001. Available: http://www.sciencedirect.com/science/article/pii/S2210650212000363
39. Popovici E, Bucci A, Wiegand RP, De Jong ED (2012) Coevolutionary principles. In: Rozenberg G, Bäck T, Kok JN (eds) Handbook of natural computing. Springer, Heidelberg. https://doi.org/10.1007/978-3-540-92910-9_31
40. Hillis WD (1990) Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D Nonlinear Phenom 42(1–3):228–234. https://doi.org/10.1016/0167-2789(90)90076-2. Available: http://www.sciencedirect.com/science/article/pii/0167278990900762

41. Potter MA, Jong KA (2000) Cooperative coevolution: an architecture for evolving coadapted subcomponents. Evol Comput 8(1):1–29. https://doi.org/10.1162/106365600568086. Available: http://www.mitpressjournals.org/doi/abs/10.1162/106365600568086
42. Yang Z, Tang K, Yao X (2008) Large scale evolutionary optimization using cooperative coevolution. Inf Sci 178(15):2985–2999. https://doi.org/10.1016/j.ins.2008.02.017. Available: http://www.sciencedirect.com/science/article/pii/S002002550800073X
43. Urbanowicz RJ, Moore JH (2009) Learning classifier systems: a complete introduction, review, and roadmap. J Artif Evol Appl 2009:1–25
44. Ochoa G, Harvey I, Buxton H (1999) On recombination and optimal mutation rates. In: Proceedings of genetic and evolutionary computation conference, vol 1, pp 488–495. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.50.2369
45. Eiben AE, Smit SK (2011) Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm Evol Comput 1(1):19–31. https://doi.org/10.1016/j.swevo.2011.02.001. Available: http://www.sciencedirect.com/science/article/pii/S2210650211000022
46. Fogel LJ (1962) Autonomous automata. Ind Res 4(2):14–19
47. Ochoa G, Blum C, Chicano F (2015) Evolutionary computation in combinatorial optimization. Springer International Publishing: Imprint: Springer, Cham
48. Bajpai RP (ed) (2014) Innovative design, analysis and development practices in aerospace and automotive engineering: I-Dad 2014, 22–24 Feb 2014. Springer Science & Business, Singapore
49. Gaurav A, Kumar V, Nigam D (2012) New applications of soft computing in bioinformatics: a review. J Pure Appl Sci Tech 11(1):12–22
50. Gupta SK, Ramteke M (2014) Applications of genetic algorithms in chemical engineering II: case studies. In: Applications of metaheuristics in process engineering. Springer, Cham, pp 61–87
51. Bentley P, Corne D (2002) Creative evolutionary systems. Morgan Kaufmann, San Francisco
52. Chen SH (ed) (2012) Genetic algorithms and genetic programming in computational finance. Springer Science & Business Media, New York
53. Gen M, Cheng R (1996) Genetic algorithms and manufacturing systems design, 1st edn. Wiley, New York
54. Adeli H, Sarma KC (2006) Cost optimization of structures: fuzzy logic, genetic algorithms, and parallel computing. Wiley, Chichester
55. Lones MA, Tyrrell AM (2007) A co-evolutionary framework for regulatory motif discovery. https://doi.org/10.1109/CEC.2007.4424978
56. Lones M, Alty JE, Lacy SE, Jamieson DR, Possin KL, Schuff N, Smith SL (2013) Evolving classifiers to inform clinical assessment of parkinson's disease. In: 2013 IEEE symposium on computational intelligence in healthcare and e-health (CICARE), pp. 76–82. IEEE
57. Lones M, Turner AP, Caves LS, Stepney S, Smith SL, Tyrrell AM (2014) Artificial biochemical networks: evolving dynamical systems to control dynamical systems. IEEE Trans Evol Comput 18(2):145–166
58. Lones MA, Smith SL, Tyrrell AM, Alty JE, Jamieson DS (2013) Characterising neurological time series data using biologically motivated networks of coupled discrete maps. BioSystems 112(2):94–101