Genetic Algorithms

15

Carlos García-Martínez, Francisco J. Rodriguez, and Manuel Lozano

Contents

C. García-Martínez (✉)
Department of Computing and Numerical Analysis, University of Córdoba, Córdoba, Spain
e-mail: cgarcia@uco.es

F. J. Rodriguez and M. Lozano
Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain
e-mail: fjrodriguez@decsai.ugr.es; lozano@decsai.ugr.es

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_28

Abstract

This chapter presents the fundamental concepts of genetic algorithms (GAs) that have become an essential tool for solving optimization problems in a wide variety of fields. The first part of this chapter is devoted to the revision of the basic components for the design of GAs. We illustrate this construction process through its application for solving three widely known optimization problems as knapsack problem, traveling salesman problem, and real-parameter optimization. The second part of the chapter focuses on the study of diversification techniques that represent a fundamental issue in order to achieve an effective search in GAs. In fact, analyzing its diversity has led to the presentation of numerous GA models in the literature. Similarly, the hybridization with other metaheuristics and optimization methods has become a very fruitful research area. The third part of the chapter is dedicated to the study of these hybrid methods. In closing, in the fourth part, we outline the wide spectrum of application areas that shows the level of maturity and the wide research community of the GA field.

Keywords

Genetic Algorithms · basic components · GA design · population diversity · diversity maintenance · diversity generation · hybrid genetic algorithms

Introduction

Genetic algorithms, or GAs, are nowadays an important field in artificial intelligence and operations research, with more than 2,500 publications per year in the last 10 years. Contrary to their most common type of application, GAs were not initially presented in the 1970s for problem solving, but as an instrument for simulating the biological evolution of adaptive natural systems [58]. Nevertheless, it was precisely the simulated adaptive ability that promoted, in the subsequent years, the opportunity for tackling optimization problems successfully [26, 27, 43]. In this chapter, we focus on their utility for problem solving, so GAs can operationally be described as iterative algorithms that sample candidate solutions for a given problem, with the aim of reaching the best configurations.

GAs take their inspiration from the biological evolution of species. Thus, the following ideas belong to their core foundations:

- Species evolve in populations of individuals that compete for survival within an environment.
- New individuals appear because of the recombination of previous individuals.
- Darwin's theory of *natural selection* establishes that the fittest individuals, those better adapted to the environment, get higher chances of surviving and breeding.

Thus, less fitted individuals may probably perish at some point of the evolution process.
- Individuals' adaptation to the environment strongly depends on their phenotypic characteristics, which, in turn, are principally determined by their genotype.
- Individuals might eventually suffer mutations at their genotype level, which may affect their phenotype and adequacy to the environment.

Using these ideas, the following principles govern most of GAs for optimization:

- The environment is defined by the problem to be addressed.
- Individuals, also known as *chromosomes*, represent candidate solutions for the problem.
- Their *genotypes* encode the candidate solutions for the problem. The genotype-phenotype translation establishes how the chromosomes should be interpreted to get the actual candidate solutions.
- The *fitness* of individuals depends on their adequacy on the given problem, so fitter individuals are more probably to survive and breed.
- There is an evolving *population of individuals*, where new individuals may get into and other may perish.
- New individuals are generated as the consequence of the *recombination* and/or the *mutation* of previous ones.

It should be pointed out that similar approaches were being developed since the 1960s, and, thus, there are nowadays several optimization methods sharing a reasonable resemblance. All these approaches have been gathered up under the term *evolutionary algorithms* [29, 37], pointing out the fact that these strategies work with candidate solutions which evolve toward better configurations. However, a distinguishing feature of GAs at the time was the concept of recombination implemented as a crossover operator. This, in contrast with the neighborhood exploration of just one solution, carried out by the existing heuristics and by the operators of other algorithms, introduced the idea of combining the information from two candidate solutions to generate a new one. Years afterward, the idea of recombining two or more solutions was adopted by most representatives of evolutionary computation.

The main aim of this chapter is to introduce the general and practical guidelines for designing GAs for given problems. Thus, we will gather up classical ideas from their origins and the latest tendencies without going into the specific and theoretical details. The interested reader is referred to [95] for a survey of the works developed in this regard.

The chapter is organized as follows. GA Construction Section offers several hints to newcomers for the design and implementation of a GA. Three different and widely known classes of optimization problems are used for reviewing the basic GA components, and several classic alternatives are provided for each of them. Diversification Techniques for GAs Section analyzes the premature convergence problem of GAs produced by the lack of diversity and presents different methods and operators

that attempt to favor the diversity of the population in GAs. Hybrid GAs Section is devoted to another research area with an important number of algorithms reported in the last years and that is intended to exploit the complementary character of different optimization strategies. Finally, GA Applications Section outlines the application areas of GAs and the major challenges faced by GAs to deal with optimization problems in these areas.

GA Construction

In this section, we revise the basic ingredients for designing a GA. We comment some of the original and widely used strategies for each of these components without getting into a deep discussion whether they perform better or worse than other approaches.

Given an optimization problem with a set of decision variables arranged in a vector $\mathbf{x}$, let $f : X \rightarrow \Re$ be a function that maps any configuration of the decision variables to a quality value. The aim is to find the combination of values for the decision variables that maximizes, or minimizes, function f. This function is usually known as the objective function. We assume that function f is to be maximized for the rest of the chapter, being trivial most of the needed modifications for minimization problems. In some cases, there are additional restrictions that solutions must satisfy, which are modeled by a set of inequality and equality constraints, $g_j(\mathbf{x}) \leq 0$ and $h_j(\mathbf{x}) = 0$ with $j = 1, \ldots, M$, respectively. We will use the following widely known optimization problems throughout the chapter to illustrate the application of GAs to problems with very different characteristics:

Knapsack Problem: In the classic knapsack problem [22], we are given a set of N objects with profits $p_i \geq 0$ and weights $w_i \geq 0$ and a container, the knapsack, with a maximum capacity C. The goal is to decide which objects should be included in the knapsack in such a way that the sum of the profits of the selected objects is maximized and the capacity C is not exceeded by the sum of their respective weights. Note that the decision variables of this problem should indicate which objects are included into the knapsack and which are not. Likewise, given a valid candidate solution that satisfies the capacity constraint ($g(\mathbf{x}) = -C + \sum_{i \in \text{Knapsack}} w_i$), the objective function f is the sum of the profits p_i of those objects in the knapsack.

Traveling Salesman Problem: This problem [50] consists in the design of travel plans visiting a given set of cities or locations so that the traveled distance is minimized. The information given is the set of cities, all of them have to be visited, and the distances between each pair of cities. In this case, decision variables should represent the order in which the cities are visited, and the objective function is the sum of the distance between consecutive cities plus the journey back to the first city from the last one.

Real-Parameter Optimization: Here, we are given an objective function whose decision variables take values from the continuous space $\Re$. In the context of optimization with metaheuristics, this objective function is treated as a *black box* so analytical methods cannot be applied. This is often the case when the evaluation of candidate solutions involves the execution of an external simulation procedure, such as wind tunnels or driving simulators, whose internal processes are obscure or they are not subject of analysis. The distinguishing feature of these problems, with regard to combinatorial problem, is that decision variables take real values instead of binary or integer ones.

Whichever is the problem, the design of a GA goes throughout the following steps and not necessarily in this order:

Representation

The practitioner needs to specify how the decision variables of the problem are represented in the computer program to allow the GA to generate the different possible configurations. In particular, the GA community stresses the difference between the solution representation, *genotype*, from arrays of binary, integer (or elements from an alphabet), or floating-point variables to complex structures such as trees, graphs, or objects of specifically designed classes and the solution which is actually represented, *phenotype*. Consider the previous optimization problems:

Knapsack Problem – Representation: The classic representation for this problem is the *binary coding*, which is actually the traditional representation in GAs regardless the problem. Solutions are represented by an array of N binary variables, as much as objects in the problem, where the value 1 at the i-th position indicates that the i-th object should be included into the knapsack and 0 means the opposite. Therefore, the following example indicates that we should select the first, third, fourth, and seventh objects and discard the others.

$$[\,1\,0\,1\,1\,0\,0\,1\,]$$

Here, the array of binary variables is the genotype, and the interpretation of that genotype, the actual configuration of the knapsack, is the phenotype. However, note that we could have adopted a different representation such as a variable-length array or set with the integer indexes of the included objects ([1 3 4 7] in the case above). We assume from now on that binary coding is adopted for this problem.

Traveling Salesman Problem – Representation: Given that the important information here is the order in which cities are visited, a natural way to represent solutions is by means of *permutations* [72], such as the following one:

$$[\,3\,4\,5\,7\,2\,6\,1\,]$$

These permutations are the genotype from which different interpretations could be developed, the phenotype. The most commonly used is the one that establishes traveling relations between consecutive values, i.e., in the example above, the agent travels from the third city to the fourth one, from the fourth to the fifth, and so on; finally, the agent returns to the third city from the first one (last value in the solution). As previously, other coding schemes could be adopted, such as the binary encoding of each of the integers in the example ([011 100 101 111 010 110 001]). Particularly, an indirect technique with relative success in the field is the *random key encoding* [48]. This represents a permutation as the sorting order of an array of numbers. Therefore, the array [3.0 4.1 0.2 0.3 −0.2 7 1] would be first translated into the permutation [5 3 4 7 1 2 6], from which the appropriate interpretation should be extracted. For the rest of the chapter, we adopt the former representation, which indicates the order by which cities are visited.

Real-Parameter Optimization – Representation: Initially, real (and integer) optimization problems were addressed with the binary coding scheme, considering either a natural or a gray transformation [16,43,58]. This latter intends to minimize the number of hamming differences between close final interpretations. However, real encoding is much widely used lately for real-parameter optimization problems, which avoids the differentiation between genotype and phenotype [55,61,81]:

$$[\,0.5 \;\; 0.0001 \;\; 0.2342 \;\; 1.0 \;\; 100.0 \;\; 0.7 \;\; -7.2\,]$$

Initial Population: Generating Candidate Solution from Scratch

At the beginning of the search process, GAs conform an initial population of candidate solutions. Usually, there is not much information to be exploited to generate these candidate solutions and the following alternatives are commonly applied:

- *Random solutions*: When there is no other information apart from the considered representation (binary, integer, permutations, real, trees, etc.), one of the easiest and most widely used methodologies is to generate the initial population at random. Thus, the task of generating the initial population becomes the one of producing arrays of random binary, integer, or floating-point numbers, trees, graphs, or a mixture of them.
- *Diverse solutions*: As we will see later, the practitioner is often concerned with the presence of diverse solutions in the population. For the case of the initial population, many works recommend random procedures that incorporate some bias for generating diverse solutions and, thus, covering the search more uniformly [91]. One simple technique is a generalization of the Latin hypercube [60] which works as follows. Suppose that N initial solutions should be generated, whose genotype is an array with l variables that take values from an m-ary alphabet. First, l random permutations of $\{1, \ldots, N\}$ are composed.

Table 1 Latin hypercube sampling with $N = 6$, $m = 3$, and $l = 7$

Individual	P_1	P_2	P_3	P_4	P_5	P_6	P_7	Genotype
I_1	2	5	5	6	2	3	2	[3 3 3 1 3 1 3]
I_2	5	2	2	4	1	6	6	[3 3 3 2 2 1 1]
I_3	6	3	6	2	5	4	4	[1 1 1 3 3 2 2]
I_4	4	1	1	3	4	2	5	[2 2 2 1 2 3 3]
I_5	1	4	4	5	6	5	1	[2 2 2 3 1 3 2]
I_6	3	6	3	1	3	1	3	[1 1 1 2 1 2 1]

Then, solutions are generated concatenating the corresponding values of the permutations modulo m (plus 1 if needed), i.e., the first solution considers the first value of the permutations and so on. See an example in Table 1. Note that N should be a multiple of m to assure a uniform sample of the m values. In case of using a real encoding scheme, one possibility is to divide the domain of the variables in m disjoint intervals. The previous procedure is applied and the variable finally takes a uniform random value from the selected interval.

- *Heuristic solutions*: In case you have some information of the problem at hand, you may exploit it to sample initial solutions better than those randomly generated. For example, in the knapsack problem, objects with better weight-profit ratios can be given higher probabilities of being included into the knapsack; or in the traveling salesman problem, the nearest neighbor heuristic can be randomized to generate different solutions [115].

In [65], different population initialization techniques are reviewed and categorized into a taxonomy.

Evaluation

Solutions are evaluated to decide whether they will, or will not, take part in the evolution process. This means assigning a fitness value to these candidate solutions. The most common practice is to assign a fitness value which is computed exactly as, or directly from, the objective function $f(\cdot)$. For example, the sum of the profits of the included objects can be used in the knapsack problem or the traveled distance, in the traveling salesman problem. In some other occasions, you may be interested in another computation such as a scaling procedure to avoid the influence of the scale of the function, or including other information such as diversity maintenance or constraint violation, often addressed with penalty functions [100, 118].

A very different strategy is to consider a relative measure such as the ranking of the chromosomes in fitness order [94, 113]. Though there is information loss, a clear advantage is that fitness values do not depend on the scale of the objective function.

Selection Operator

This operator is traditionally the only one in charge of assigning more reproduction possibilities to better solutions and, therefore, biasing the search process toward the pursued objective. GAs apply this operator to select those individuals allowed to breed and generate new candidate solutions.

The first selection operator applied is known as the *roulette-wheel* method. The idea is to assign selection probabilities, or roulette portions, to the solutions in the population, which are proportional to their fitness values. Thus, better individuals get higher probabilities of being selected. Then, the roulette is drawn as many times as the number of individuals needed. An enhanced method is the *stochastic universal selection* of Baker [9]. In this case, the roulette has multiple equally spaced spinners, as much as individuals need to be selected, and is drawn just once. In contrast to roulette wheel, Baker's method reduces the stochastic possibility of selecting the best individual too many or not enough times.

Tournament is another selection operator widely applied nowadays because of its simplicity [47]. This method composes a set of t_{size} random individuals from the population, and the best one is selected for breeding. Multiple tournaments are simulated to get the parent population. Note that parameter t_{size} controls the bias toward the best solutions, being $t_{\text{size}} = 2$ a classic setting, i.e., binary tournament.

Genetic Operators

Crossover and mutation operators are the GA components that generate new candidate solutions, which are afterward evaluated and considered for the subsequent evolution iteration. Between these two, the crossover has traditionally got a superior significance, and mutation often acts in the background, being applied less frequently. The goal of the former is to combine the information of two or more parent chromosomes from the current population, while the second usually modifies a chosen chromosome random and locally.

There is not a clear methodology for the order in which these operators should be applied. Commonly, two GA parameters govern the frequency of crossover and mutation applications, namely, crossover and mutation probabilities, and the process is often carried as follows. The selected parent solutions firstly undergo crossover according to the crossover probability, and offspring is transferred to the mutation step; parents are otherwise transferred to the mutation phase. Subsequently, mutation is applied on the received solutions according to the mutation probability, and the result is temporarily stored for the next generation.

Given the strong interaction between the solution representation and genetic operators, we revise some classical proposals in the context of previous problems.

Crossover Operator

We show in this section some examples for combining the information from more than one parent through crossover application.

Table 2 Example of binary crossover operators with parents $P_1 = [\,0\,1\,1\,0\,1\,0\,0\,]$ and $P_2 = [\,1\,1\,1\,1\,0\,0\,1\,]$

Name	Intermediate steps			Result
One point Crossover	Cut point 4	Heads [0 1 1 0 [1 1 1 1	Tails 1 0 0] 0 0 1]	Offspring [0 1 1 0 0 0 1] [1 1 1 1 1 0 0]
Uniform Crossover		Genes from P_1 1, 3, 5, 6	Genes from P_2 2, 4, 7	Offspring [0 1 1 1 1 0 1]
Half uniform Crossover	Differences 1, 4, 5, 6	Genes from P_1 4, 5	Genes from P_2 1, 6	Offspring [1 1 1 0 1 0 1]

Knapsack Problem – Crossover Operator: The firstly proposed crossover operator took its inspiration from biology and is widely known as *one-point crossover* operator. Given two binary strings representing the corresponding parent solutions, this operator chooses a random position (4 in the example in Table 2) and combines the head of the first parent with the tail of the second one, generating a new candidate solution, and vice versa for a second candidate solution. However, other crossover operators such as uniform crossover [102] and half-uniform crossover [33] (see also The Crossover Operator as Diversification Agent Section) usually perform better. Both create a new offspring by selecting randomly each of the gene values from the parents, but the latter assures that half of the number of differences are taken from one parent and the other half from the other parent. In the particular case of the knapsack problem, the resulting solutions may violate the capacity constraint, so penalty functions or repair operators would be needed.

Traveling Salesman Problem – Crossover Operator: Problems using the permutation coding, such as this one, require crossover operators to accept permutations as input and produce valid permutations as output. Note that previous operators do not satisfy this necessity. Therefore, specific operators are proposed for this type of problems (see Table 3). The firstly proposed one was *partially mapped crossover* [45], which is aimed precisely at generating sequences of numbers without repetition. Its idea is to divide the two parent permutations in three segments each according to two randomly chosen positions (2–4 in Table 3). Then, the inner segment of one parent is combined with the outer ones of the other, and repeated genes in the outer segments are changed according to the mapping rules defined in the inner segment. *Order crossover* [23] is another option which intends to exploits the relative order of the gene values instead of their absolute positions, as previous operator does. Firstly sampling two random positions, this operator copies the inner segment into the offspring and continues copying those valid gene values from the secondly sampled position of the other parent (see Table 3). Given that different operators focus on different interpretations of the information transmitted, the practitioner should identify what is the one most appropriate for the adopted problem representation.

Table 3 Example of crossover operators for permutations with parents $P_1 = [\,1\,2\,3\,4\,5\,6\,7\,]$ and $P_2 = [\,2\,4\,5\,7\,1\,6\,3\,]$

Name	Intermediate steps			Result
Partially	Cross. points	Comb. and rep. removal	Mappings	Offspring
Mapped	2–4	[2 | 2 3 4 | 1 6 3]	2 → 4, 3 → 5, 4 → 7	[7 | 2 3 4 | 1 6 5]
Crossover		[1 | 4 5 7 | 5 6 7]	4 → 2, 5 → 3, 7 → 4	[1 | 4 5 7 | 3 6 2]
Order	Cross. points	Inner segment	Valid values	Offspring
Crossover	4–6	[− − − | 4 5 6 | −]	[2 4 | 5 7 1 6 | 3]	[2 7 1 | 4 5 6 | 3]
		[− − − | 7 1 6 | −]	[1 2 | 3 4 5 6 | 7]	[3 4 5 | 7 1 6 | 2]

Table 4 Example of real-parameter crossover operators with parents $P_1 = [\,0.3\;\;7\,]$ and $P_2 = [\,0.4\;\;1\,]$, $\alpha = 0.5$, and domains $x_1 \in [0, 1]$ and $x_2 \in [-10, 9]$

Name	Intermediate steps			
BLX-α	$I_i^a \mid p_i^1 - p_i^2 \mid$	Lower bound	Upper bound	Interval
		$\max(l_i, \min(p_i^1, p_i^2) - \alpha I_i^a)$	$\min(u_i, \max(p_i^1, p_i^2) + \alpha I_i^a)$	
o_1	0.1	$\max(0,\ 0.3 - 0.05)$	$\min(1,\ 0.4 + 0.05)$	[0.25, 0.45]
o_2	6	$\max(-10,\ 1 - 3)$	$\min(9,\ 7 + 3)$	[−2, 9]
PBX-α	$I_i^a \mid p_i^1 - p_i^2 \mid$	Lower bound	Upper bound	Interval
		$\max(l_i,\ p_i^1 - \alpha I_i^a)$	$\min(u_i,\ p_i^1 + \alpha I_i^a)$	
o_1	0.1	$\max(0,\ 0.3 - 0.05)$	$\min(1,\ 0.3 + 0.05)$	[0.25, 0.35]
o_2	6	$\max(-10,\ 1 - 3)$	$\min(9,\ 1 + 3)$	[−2, 4]

Real-Parameter Optimization – Crossover Operator: Similar to previous cases, there are many proposals for the combination of the information in two real-coded parent solutions. In this case, researchers pay attention to the capacities of the crossover to converge, expand, or even correlate the changes on the decision variables [56]. An interesting characteristic of most real-parameter crossover operator is whether they tend to sample new solutions near the centroid of the parents or near the parents. BLX-α [34] is an example of centroid-centered operator and PBX-α is its parent-centered version [39,76]. For each offspring gene o_i, both operators create an interval $I = [a, b]$ from which the actual value is uniformly and randomly drawn. In both cases, the amplitude of the interval depends on the difference between the parent gene values, $I_i^a = \text{abs}(p_i^1 - p_i^2)$, and the domain of the variable, $[l_i, u_i]$. The difference is the location of that interval, biased toward the average of previous parent gene values, in the case of BLX-α, or the value of the first parent, PBX-α. Table 4 shows an example of how both operators compute the intervals from where offspring gene values are sampled.

Mutation Operator

As for the crossover operator, mutation is strongly connected with the adopted representation. Therefore, we will see some simple examples for any of the three problems considered through this chapter. Apart, we shall mention that the

community usually distinguishes between applying mutation at individual or gene level. In the first case, the chromosome is mutated only once according to the mutation probability. One gene is randomly selected and modified, which could involve additional modifications in one or a few other genes. In the latter, all the genes of the chromosome are visited, and each one is mutated according to the mentioned probability, so chromosomes might suffer stronger alterations. In this case, the mutation probability is often much smaller.

Knapsack Problem – Mutation Operator: Flipping is the simplest mutation operator for binary strings. If mutation applies at the individual level, a random gene is selected and its value is changed, from 0 to 1 or the other way around. If mutation is at the gene level, each variable is visited once and flipped according to the mutation probability. One should notice that other operators might be more appropriate for the problem at hand. For example, in the case of the knapsack problem, flipping one bit from 0 to 1 might leave room for inserting another object in the knapsack, and changing one bit from 1 to 0 may make the solution unfeasible. Thus, you might prefer using another more specialized operator, such as flipping two bits with different values, instead of this general one.

Traveling Salesman Problem – Mutation Operator: Assigning random values to one or several genes is often applied in integer-coded problems; however, this strategy is not appropriate for permutation codings. Instead, mutation operators usually apply swappings, insertions, reorderings, or sublist inversions [29, 98]. Table 5 shows the application of swap [11, 88] and sublist inversion [58]. In both cases, two positions are randomly drawn. Notice, however, that sublist inversion modifies less arcs of the solution path in symmetric graphs than swapping the selected positions. This is due to the fact that traveling direction does not affect the solution cost in this kind of graphs. Thus, sublist inversion is often preferred for this problem.

Real-Parameter Optimization – Mutation Operator: Uniform and norm random variables, with a symmetric distribution around zero, are commonly used to perturb a given real-coded solution locally. In contrast to previous cases, it is usual to modify more than one and often all the variables of the solutions at the same time, by adding up a complete randomly drawn vector. Altering either one or all the variables, practitioners often pay attention at the global impact of the

Table 5 Example of mutation operators for permutations with chromosome $S = [\,2\,4\,5\,7\,1\,6\,3\,]$

Swap	Mut. points 2–5	Result [2 | 1 | 5 7 | 4 | 6 3]	Arcs removed $2 \to 4,\ 4 \to 5,$ $7 \to 1,\ 1 \to 6$	Added arcs $2 \to 1,\ 1 \to 5$ $7 \to 4,\ 4 \to 6$
Sublist Inversion	Mut. Points 2–5	Result [2 | 1 7 5 4 | 6 3]	Arcs removed $2 \to 4,\ 1 \to 6$	Added arcs $2 \to 1,\ 4 \to 6$

mutation so that the difference between the original and the mutated solution, under the Euclidean metric, for instance, is controlled. A widely used operator is the nonuniform mutation [61, 81], which reduces the intensity of mutation with the number of generations. Particularly, a variation vector $\Delta = [\ \delta_1\ \ldots\ \delta_i\ \ldots\ \delta_N\ ]$, computed as follows, is added up to the given solution $X = [\ x_1\ \ldots\ x_i\ \ldots\ x_L\ ]$:

$$\delta_i = \begin{cases} (u_i - x_i)(1 - z_i)^{\gamma}, & \text{with probability } 1/2 \\ (l_i - x_i)(1 - z_i)^{\gamma}, & \text{otherwise} \end{cases}$$

$$\gamma = \left(1 - \frac{t}{t_{\max}}\right)^{\beta}$$

where z_i is an uniform random variable in $[0, 1]$, u_i and l_i are the box bounds of variable x_i, β is a parameter with $\beta > 0$, and t and $t_{\max}$ are, respectively, the current and maximal number of generations.

Evolution Model and Replacement Strategy

The traditional GA applied a *generational* evolution strategy. This means that evolutions occur at generations where a complete new population is generated from the current one, which, in turn, becomes the current population for the next generation. The pseudocode of this generational GA is shown in Fig. 1. A possible undesirable effect of this model is that the new population might be worse, in quality terms, than the current population. So usually, the best solution from the current population is artificially inserted in the new population, particularly when there is not any better solution in the new population. This evolution model is known as *generational with elitism* [10].

Another commonly applied model is the *steady-state GA* [28], which contrarily generates a reduced set of new chromosomes, usually just one solution, that compete to enter into the population (see pseudocode in Fig. 2). In steady-state GAs, the designer has to specify a replacement policy, which determines whether the new solutions enter the population or are discarded and which solution from the population is removed to make room for the new one, if this is accepted. A simple policy to introduce the new solution into the population is replacing the worst solution if the new one is better. Other strategies consider diversity measures, like replacing the most similar solution [52] (see also Strategies to Maintain Diversity Section, crowding methods) or the worst parent.

Stop Condition

GAs evolve until a certain stopping criterion is met. Practitioners commonly consider a maximum number of generations, fitness evaluations, or a maximal processing time. These criteria are widely used when one is interested in comparing

Input:

NP: Population size

p_c: Crossover probability

p_m: Mutation probability

Output:

S: Best solution found

```
//Initial population
for i=1 to NP do
    p_i ← Generate solution;
    p_i.fitness ← Evaluate(p_i); //acc. to objective function f(·)
end
//Evolution
repeat
    P_0 ← ∅;
    //Offspring generation
    repeat
        parents ← Selection(P);
        offspring ← Crossover(parents); //acc. to p_c
        offspring ← Mutation(offspring); //acc. to p_m
        offspring.fitness ← Evaluate(offspring);
        P_0 = P_0 ∪ offspring;
    until |P_0| = NP;
    P ← P_0;
until Stop-condition is met;
return Best generated solution;
```

Fig. 1 Basic scheme of a generational GA

```
Input:
 NP: Population size
 pc: Crossover probability, usually equal to 1 in steady-state GAs
 pm: Mutation probability
Output:
 S: Best solution found
//Initial population
for i=1 to NP do
    pi ← Generate solution;
    pi.fitness ← Evaluate(pi); //acc. to objective function f(·)
end
//Evolution
repeat
    parents ← Selection(P);
    offspring ← Crossover(parents);
    offspring ← Mutation(offspring); //acc. to pm
    offspring.fitness ← Evaluate(offspring);
    P ← Replacement(offspring, P);
until Stop-condition is met;
return Best generated solution;
```

Fig. 2 Skeleton of a steady-state GA

the performance of several algorithms, and the same computational resources have to be provided for a fair study.

On other occasions, especially when one is aimed at solving a particular problem, GAs are executed until a minimal solution quality is attained or the population converges so further progress is very difficult. In these cases, practitioners often need to be able to consult the progress of the search to decide at the moment, whether continuing the evolution or truncating it and applying a restart procedure (see also section "Methods to Generate Raw Diversity").

Diversification Techniques for GAs

There are two primary factors in the search carried out by a GA [20]: population diversity and selective pressure. In order to have an effective search, there must be a search criteria (the fitness function) and a selection pressure that gives individuals with higher fitness a higher chance of being selected for reproduction, mutation, and survival. Without selection pressure, the search process becomes random, and promising regions of the search space would not be favored over regions offering no promise. On the other hand, population diversity is crucial to a GAs ability to continue the fruitful exploration of the search space. If the lack of population diversity takes place too early, a premature stagnation of the search is caused. Under these circumstances, the search is likely to be trapped in a region not containing the global optimum. This problem, called premature convergence, has long been recognized as a serious failure mode for GAs [53, 119]. In the literature, many approaches have been proposed to introduce new methods and operators that attempt to favor the diversity of the population to overcome this essential problem of genetic algorithms. Next, we present a quick overview of them.

Methods to Generate Raw Diversity

Premature convergence causes a drop in the GAs efficiency; the genetic operators do not produce the feasible diversity to tackle new search space zones, and thus the algorithm reiterates over the known zones producing a slowing-down in the search process. Under these circumstances, resources may be wasted by the GA searching an area not containing a solution of sufficient quality, where any possible improvement in the solution quality is not justified by the resources used. Therefore, resources would be better utilized in restarting the search in a new area, with a new population. This is carried out by means of a restart operator [42, 44], which introduces chromosomes with high *raw diversity* to increase the average diversity level, thus to ensure the process can jump out the local optimum and to revolve again.

Eshelman's CHC algorithm [32] represents a GA with elitist selection and a highly disruptive recombination operator which restarts the search when the population diversity drops below a threshold level. The population is reinitialized by using the best individual found so far as a template for creating a new population. Each individual is created by flipping a fixed proportion (35%) of the bits of the template chosen at random without replacement. If several successive reinitializations fail to yield an improvement, the population is completely (100%) randomly reinitialized.

Another GA that utilizes population reinitialization is the micro GA [43]. In general, a micro GA is a small population GA which evolves for many generations. When after a number of generations the micro GA population converges, the evolutionary process is reinitialized by preserving the best individual and substituting the

rest of the population with randomly generated individuals. The first implementation of a micro GA was reported by Krishnakumar [69], who used a population size of five individuals, tournament selection, single-point crossover with probability, elitism, and restart operator. The population was considered converged when less than 5% of the population bits were different from the bits of the best individual.

A recent GA model, called saw-tooth GA [68], manages a variable population size with periodic reinitialization following a saw-tooth scheme with a specific amplitude and period of variation. In each period, the population size decreases linearly, and at the beginning of the next period, randomly generated individuals are appended to the population.

Strategies to Maintain Diversity

Pioneer works on the way diversity may be retained throughout the GA run focused on the design of alternative selection mechanisms. For example, in the linear ranking selection [8], the chromosomes are sorted in order of raw fitness, and then the selection probability of each chromosome is computed according to a linear function of its rank. With this selection mechanism, every individual receives an expected number of copies that depends on its rank, independent of the magnitude of its fitness. This may help prevent premature convergence by preventing super individuals from taking over the subpopulations within a few generations.

Disruptive selection [70] attempts to accomplish this objective as well. Unlike conventional selection mechanisms, this approach devotes more trials to both better and worse solutions than it does to moderate solutions. This is carried out by modifying the objective function of each chromosome, C, as follows: $f'(C) = |f(C) - \bar{f}|$, where $\bar{f}$ is the average value of the fitness function of the individuals in the population. A related selection method is the fitness uniform selection scheme (FUSS) [59]. FUSS generates selection pressure toward sparsely populated fitness regions, not necessarily toward higher fitness. It is defined as follows: if $f_{\min}$ and $f_{\max}$ are the lowest and highest fitness values in the current population, respectively, we select a fitness value uniformly in the interval $[f_{\min}, f_{\max}]$. Then, the individual in the population with fitness nearest to this value is selected. FUSS results in high selection pressure toward higher fitness if there are only a few fit individuals, and the selection pressure is automatically reduced when the number of fit individuals increases. In a typical FUSS population, there are many unfit and only a few fit individuals. Fit individuals are effectively favored until the population becomes fitness uniform. Occasionally, a new higher fitness level is discovered and occupied by a new individual, which then, again, is favored. Finally, another technique being worthy of mention is the repelling algorithm [114], which modifies the fitness function to increase the survival opportunity of chromosomes with rare alleles.

There are different replacement strategies for steady-state GAs that try to maintain population diversity as well. In [77], the authors propose a replacement strategy that considers two features of the individual to be included into the

population: a measure of the contribution of diversity to the population and the fitness function. It tries to replace a chromosome in the population with worst values for these two features. In this way, the diversity of the population increases and the quality of the solutions improves, simultaneously. The goal of this strategy is to protect those individuals that preserve the highest levels of useful diversity.

Other replacement methods to promote population diversity are the crowding methods [97]. They work as follows: new individuals are more likely to replace existing individuals in the parent population that are similar to themselves based on genotypic similarity. In this manner, the population does not build up an excess of similar solutions. Crowding methods promote the formation of stable subpopulations in the neighborhood of optimal solutions (niches), favoring the preservation of multiple local optima in multimodal problems. An effective crowding method is the restricted tournament selection (RTS) [52]. RTS initially selects two elements at random, A and B, from the population and perform crossover and mutation on these two elements resulting in a new element A'. Then, RTS scans ω (window size) more members of the population and picks the individual that most closely resembles A' from those ω elements. A' then competes with this element, and if A' wins, it is allowed to enter the population. Another type of crowding methods assumes that the parents would be those members of the population that are closer to the new elements. In this way, children compete with their parents to be included in the population, i.e., a family competition is held. These methods include deterministic crowding [78] and elitist recombination [106].

Some GA models were proposed in the literature that explicitly avoid the existence of duplicate individuals in the population, as a way to encourage diversity. In fact, early empirical studies confirmed that duplicate removal can enhance the performance of GA significantly [80]. For example, the non-revisiting GA [119] guarantees that no revisits ever occur in its fitness evaluations. It achieves this by interacting with a dynamically constructed binary space partitioning archive that is built up as a random tree for which its growth process reflects the evolution history of the GA and is a quick method to query whether there is a revisit. The entire previous search history is then used to guide the search to find the next unvisited position. A similar approach may be found in [51].

Finally, it is worth to mention that diploid GAs [67, 109] are evolutionary algorithms that manipulate a specific kind of diversity that becomes profitable to deal with dynamic optimization problems (in which some elements of the underlying model change over the course of the optimization). Most organisms in nature have a great number of genes in their chromosomes, and only some of the dominant genes are expressed in a particular environment. The repressed genes are considered as a means of storing additional information and providing a latent source of population diversity. Diploid GAs use diploid chromosomes which are different from natural ones in that the two strands of the diploid chromosomes are not complementary. Only some genes in a diploid chromosome are expressed and used for fitness evaluation by some predetermined dominance rules. Unused genes remain in the diploid genotype until they may later become useful (*latent diversity*).

The Crossover Operator as Diversification Agent

The mating selection mechanism determines the way the chromosomes are mated by applying the crossover to them. In the conventional GA, no mating strategy is applied to the results of selection; that is, parents are approved without any further examination after they are chosen at random or just by fitness. However, mates can be selected so as to favor population diversity [32, 107]. A way to do this is the negative assortative mating mechanism. Assortative mating is the natural occurrence of mating between individuals of similar genotype more or less often than expected by chance. Mating between individuals with similar genotype more often is called positive assortative mating and less often is called negative assortative mating. Fernandes et al. [35] assume these ideas in order to implement a parent selection mechanism for the crossover operator. A first parent is selected by the roulette wheel method, and n_{ass} chromosomes are selected with the same method. Then, the similarity between each of these chromosomes and the first parent is computed. If assortative mating is negative, then the one with less similarity is chosen. If it is positive, the genome that is most similar to the first parent is chosen to be the second parent. Clearly, the negative assortative mating mechanism increases genetic diversity in the population by mating dissimilar genomes with higher probability.

The crossover operator has always been regarded as one of the main search operators in GAs [66] because it exploits the available information in previous samples to influence future searches. This is why research has been focused on developing crossover operators with an active role as effective diversification agents. The half-uniform crossover used in the CHC algorithm [32] is a highly disruptive crossover that crosses over exactly half of the nonmatching alleles (the bits to be exchanged are chosen at random without replacement). This way, it guarantees that the two offspring are always at the maximum Hamming distance from their two parents, thus proposing the introduction of a high diversity in the new population and lessening the risk of premature convergence. It is worth of mention that CHC applies this operator along with (1) a reproduction restriction that assures that selected pairs of chromosomes would not generate offspring unless their Hamming distance is above a certain threshold and (2) a conservative selection strategy with high selective pressure (it keeps the N best elements appearing so far). In fact, Kemenade et al. [110] suggest that higher selection pressures allow the application of more disruptive recombination operators.

A crossover operator that was specifically designed with the aim of diversifying the search process of the real-coded GAs is BLX-α [34] (see also Crossover Operator section where its operation is described and, moreover, an illustrative example is presented). Nomura et al. [86] provide a formalization of this operator to analyze the relationship between the chromosome probability density functions before and after its application, assuming an infinite population. They state that BLX-α spreads the distribution of the chromosomes when $\alpha > \frac{\sqrt{3}-1}{2}$ or otherwise reduces it. This property was verified through simulations. In particular, the authors observed that BLX-0.0 makes the variances of the distribution of the chromosomes

decrease, reducing the distribution, whereas BLX-0.5 makes the variances of the distribution increase, spreading the distribution.

BLX-α has a self-adaptive nature in that it can generate offspring adaptively according to the distribution of parents without any adaptive parameter [12]. BLX-α uses probability distributions that are calculated according to the distance between the decision variables in the parents. If the parents are located closely to each other, the offspring generated by this operator might be distributed densely around the parents. On the other hand, if the parents are located far away from each other, then the offspring will be sparsely distributed around them. Therefore, it may fit their action range depending on the diversity of the population by using specific information held by the parents. In this way, depending on the current level of diversity in the population, it may favor the production of additional diversity (divergence) or the refinement of the solutions (convergence). This behavior is achieved without incurring into extra parameters or mechanisms to achieve the mentioned behavior.

Diversification by Adapting GA Control Parameters

Finding robust control parameter settings (such as mutation probability, crossover probability, and population size) is not a trivial task, since their interaction with GA performance is a complex relationship and the optimal ones are problem dependent. Furthermore, different control parameter values may be necessary during the course of a run to induce an optimal exploration/exploitation balance. For these reasons, adaptive GAs [30, 54, 63, 99, 108] have been built to dynamically adjust selected control parameters or genetic operators during the course of evolving a problem solution. Their objective is to offer the most appropriate exploration and exploitation behavior.

Some adaptive techniques were presented to endow the GA with useful diversity. Specifically, the mutation probability (p_m) was considered as a key parameter to accomplish this task [89, 119]. Next, we describe different adaptive mechanisms presented in the GA literature to control this parameter.

- *Deterministic control of* p_m. A direction followed by GA research for the variation of p_m lies in the specification of an externally specified schedule which modifies it depending on the time, measured by the number of generations. One of the most considered schedules consists in decreasing p_m during the GA run [36]. For example, [57] suggested the equation $p_m = 0.1 - 0.09 \cdot \frac{g}{G}$, where g is the generation number from 1 to G. This schedule follows the heuristic "to protect the exploration in the initial stages and the exploitation later," which has been considered to design other metaheuristics, such as simulated annealing.
- *Adaptive control of* p_m. In [101], a technique for the adaptive control at individual level of p_m was proposed, in which p_m is varied depending on the fitness values

of the solutions. Each chromosome C_i has its own associated p_m value, p_m^i, which is calculated as (maximization is assumed):

$$p_m^i = \frac{f_{\max} - f_i}{f_{\max} - \bar{f}} \text{ if } f_i \geq \bar{f}, \text{ and } p_m^i = 1 \text{ if } f_i < \bar{f},$$

where f_i is the chromosome's fitness, $f_{\max}$ is the population maximum fitness, and $\bar{f}$ is the mean fitness. In this way, high-fitness solutions are protected ($p_m^i = 0$), while solutions with subaverage fitnesses are totally disrupted ($p_m^i = 1$). This technique increases p_m when the population tends to get stuck at a local optimum and decreases it when the population is scattered in the solution space.

- *Self-adaptive control of* p_m. An extra gene, p_m^i, is added to the front of each chromosome, C_i, which represents the mutation probability for all the genes in this chromosome. This gene evolves with the solution [7,108]. The values of p_m^i are allowed to vary from p_m^l to p_m^h. The following steps are considered to mutate the genes in a chromosome C_i:

  1. Apply a meta-mutation on p_m^i obtaining q_m^i. This is carried out by choosing a randomly chosen number from the interval $[p_m^i - d, p_m^i + d]$, where d is a control parameter.
  2. Mutate the genes in C_i according to the mutation probability q_m^i.
  3. Write the mutated genes (including q_m^i value) back to the chromosome.

  Crossover is presently applied only to the chromosome and has no impact on p_m^i. Each offspring resulting from crossover receives the p_m^i value of one of its parents. The initial p_m^i values are generated at random from $[p_m^l, p_m^h]$.

  The self-adaptive control of GA parameters attempts to exploit the indirect link between favorable control parameter values and fitness values, with the parameters being capable of adapting implicitly, according to the topology of the objective function [7].

Diversity Preservation Based on Spatial Separation

GA models based on the spatial separation of individuals were considered as an important way to research into mechanisms for dealing with the premature convergence problem. One of the most important representatives are the distributed GAs (DGAs) [4,31,53]. Their premise lies in partitioning the population into several subpopulations, each one of them being processed by a GA, independently of the others. Furthermore, a migration mechanism produces a chromosome exchange between the subpopulations. DGAs attempt to overcome premature convergence by preserving diversity due to the semi-isolation of the subpopulations. Another important advantage is that they may be implemented easily on parallel hardware.

Moreover, we should also point out that it is possible to improve the diversification in DGAs by designing specific migration policies [6].

Making distinctions between the subpopulations of a DGA through the application of GAs with different configurations (control parameters, genetic operators, codings, etc.), we obtain the so-called heterogeneous DGAs [53]. They are suitable tools for producing parallel multiresolution in the search space associated with the elements that differentiate the GAs applied to the subpopulations. This means that the search occurs in multiple exploration and exploitation levels. In this way, a distributed search and an effective local tuning may be obtained simultaneously, which may allow premature convergence to be avoided and approximate final solutions to be reached. An outstanding example is GAMAS [90], a DGA based on binary coding that uses four subpopulations, denoted as species I–IV. Initially, species II–IV are created. Species II is a subpopulation used for exploration. For this purpose, it uses a high mutation probability ($p_m = 0.05$). Species IV is a subpopulation used for exploitation. So, its mutation probability is low ($p_m = 0.003$). Species III is an exploration and exploitation subpopulation; the mutation probability falls between the other two ($p_m = 0.005$). GAMAS selects the best individuals from species II–IV and introduces them into species I whenever those are better than the elements in this subpopulation. The mission of species I is to preserve the best chromosomes appearing in the other species. At predetermined generations, its chromosomes are reintroduced into species IV by replacing all of the current elements in this species.

The cellular GA (cGA) [3] is another kind of decentralized GA in which the population is arranged in a toroidal grid, usually of dimension two. The characteristic feature of cGAs is a neighborhood for each individual that is restricted to a certain subset of individuals in the immediate vicinity of its position. Individuals are allowed to interact only with other individuals belonging to their neighborhood. The overlapped small neighborhoods of cGAs help in exploring the search space because the induced slow diffusion of solutions through the population provides a kind of exploration (diversification). In contrast, the exploitation (intensification) is provided inside each neighborhood to improve the quality of solutions and could be controlled by the use of appropriate genetic operators. Several studies have been carried out in order to investigate mechanisms to dynamically control the exploration/exploitation trade-off kept by cGAs. This task may be achieved through tuning the relationship between the size and/or shape of the neighborhood and the grid [2] or by adjusting the local selection method [5]. Finally, it is worth remarking that since cGAs only require communication between few closely arranged individuals, they are very suitable for a parallel implementation, as well.

Hybrid GAs

Over the last few years, a large number of search algorithms were reported that do not simply follow the paradigm of one single classical metaheuristic but they combine several components of different metaheuristics or even other

kind of optimization methods. These methods are commonly known as *hybrid metaheuristics* [14, 15, 49, 75, 103]. The main motivation behind the development of hybrid metaheuristics is to take advantage of the complementary character from a set of metaheuristics and other optimization methods to produce a *profitable synergy* [96] from their combination.

In particular, the *hybridization of GAs*, and evolutionary algorithms in general, is becoming popular due to its ability to handle several problems involving complex features such as complexity, noise, imprecision, uncertainty, and vagueness [49, 79, 92, 112]. We may also highlight that many different instantiations of hybrid GAs have been presented to solve real-world problems in many different fields such as protein structure prediction [105], image processing [13], job-shop scheduling [38], and machine learning [87], to name but a few.

The flexibility offered by the GA paradigm allows specialized models to be obtained with the aim of providing intensification and/or diversification, i.e., GAs specializing in intensification and/or diversification. The outstanding role played by GAs at present along with the great interest raised by their hybridizations with other algorithms endorse the choice of their specialist approaches as suitable ingredients to build hybrid metaheuristics with others search techniques [75].

The purpose of this section is to illustrate the different strategies used successfully in the literature to hybridize GAs and classify them according to a taxonomy based on those proposed by [103] and [93] for hybrid metaheuristics.

We firstly have split the different instances of hybrid metaheuristics into two main groups according to the architecture of the algorithms:

- *Collaborative hybrid metaheuristics*. In this category, different self-contained metaheuristics interchange information between them running sequentially or in parallel. These hybrid metaheuristics can be considered as black boxes, and the only cooperation takes place through the exchange of solutions, parameters, and so on from time to time.
- *Integrative hybrid metaheuristics*. In this case, one algorithm is in charge of the search process, whereas a subordinate method is embedded as a component of the master metaheuristic. This kind of hybridization addresses the functional composition of a single optimization method, replacing or improving a particular function within a metaheuristic with another metaheuristic.

At the same time, according to the way metaheuristics are executed, *collaborative* metaheuristics can be subdivided into two different categories:

- *Collaborative teamwork*. There are several metaheuristics that work in parallel and exchange some kind of information once in a while.
- *Collaborative relay*. Several metaheuristics are executed in a pipeline fashion. The output of each algorithm becomes the input of the next one.

Integrative hybrid metaheuristics can be also subdivided into teamwork or relay categories:

- *Integrative teamwork*. One metaheuristic (subordinate) becomes a component of another population-based metaheuristic (master) [103].
- *Integrative relay*. This kind of hybrid metaheuristics represents algorithms in which a given metaheuristic is embedded into a trajectory-based metaheuristic [103].

Collaborative Teamwork Hybrid GAs

In the case of *collaborative teamwork hybrid GAs*, we can highlight two schemes commonly used in the literature. In the first scheme, a single population is decentralized by partitioning it into several subpopulations (islands or demes), where island GAs are run performing sparse exchanges (migrations) of individuals. These kinds of models are usually referred to as distributed GAs [53] (see also Diversity Preservation based on Spatial Separation Section). Specifically, in this work several subpopulations are processed by GAs with different exploration or exploitation degrees. With this mechanism, refinement of the best solutions and expansion of the most promising zones are achieved in a parallel way. Communication between different GAs is done by sending the best solution of each population to the neighboring populations every five iterations.

In the second scheme, a set of heterogeneous metaheuristics including GAs is executed in parallel. For example, [104] combine a GA, tabu search, and a local search procedure that communicate through an adaptive memory that contains the search history. Multiple independent tabu search tasks run in parallel without any direct cooperation. However, the adaptive memory maintains a history of the search performed by these tabu search tasks. This information is used by the GA to generate individuals in unexplored regions. At the same time, elite solutions from the adaptive memory are improved with the local search procedure.

Collaborative Relay Hybrid GAs

In the *collaborative relay hybrid GAs*, a GA is executed, in a pipeline fashion, with another GA or other types of metaheuristic so that the output of each algorithm is used as input of the next one. Most instances of collaborative relay GAs follow the principle of favor exploration in the initial stages and exploitation later, inspired by the design of classical metaheuristics such as simulated annealing [75]. In this line, [17] presented an hybrid GA that combines a GA specialized for diversification and a local search process that improves the best individuals found by the GA. Whereas, [40] proposed a collaborative relay hybrid GA where the local refinement of solutions is performed by a specialized GA. They run a global real-coded GA during a determinate percentage of the available evaluations, and then they perform the local real-coded GA. The initial population for the local algorithm includes the best individuals of the global one.

Integrative Teamwork Hybrid GAs

In the case of *integrative teamwork hybrid GAs*, we can find two different approaches: *memetic algorithms* [84] and *GAs with metaheuristic-based components*. The classic scheme of memetic algorithms applies a local search method to solutions obtained by the GA that is in charge of the global search. The idea behind memetic algorithms is to provide an effective and efficient optimization method by achieving a trade-off between global exploration of GAs and local exploitation of local search procedures [83]. However, memetic algorithms also include combinations of GAs with problem-dependent heuristics, approximate algorithms, truncated exact methods, specialized recombination operators, etc. [84].

Many different instantiations of *memetic algorithms* have been presented to deal with a wide variety of application domains. However, the additional fitness function evaluations required for the local search method can increase significantly the computational cost of memetic algorithms. In order to mitigate this drawback, [83] proposed a memetic algorithm with local search chains that aims at focusing the local search action on promising areas of the search space. This is addressed by retaking the local search from solutions obtained in a previous application of the local search method, adopting also the former final strategy parameter values as its initial values.

In this line, we can find memetic algorithms in which the refinement procedure is performed by another GA instead of the classical local search procedure. With this idea, there have been presented several memetic algorithms that use GAs with a small population and short evolution or micro GAs (μGA) [64, 76, 85] (see also Diversification Techniques for GAs Section) to locally improve solutions. μGA models provide some advantages over classical local search methods. Most local search techniques find difficulties in following the right path to the optimum in complex search spaces. However, it was observed that μGAs are capable of following ridges of arbitrary direction in the search space regardless of their direction, width, or even discontinuities [64]. For example, the μGA presented in [64] is a GA with five individuals that encode perturbations. Aptitude values of these individuals depend on a solution given by the master GA. This feature ensures that search is focused on the neighborhood of the given solution, whereas low-sized population promotes high selection pressure levels.

With regard to *GAs with metaheuristic-based components*, we can find in the literature several proposals where the selection mechanism, the crossover, and the mutation operators of GAs have been replaced or extended by another classical metaheuristic. The approach proposed in [1] combines simulated annealing with a GA by extending the mutation and crossover operators with simulated annealing. Mutated and recombined solutions are accepted according to the standard simulated annealing acceptance condition. Kurahashi and Terano [71] proposed a tabu-based GA where after evaluating individuals in each iteration, they store the best individual of the generation into both long-term and short-term tabu lists. Then, the GA selection refers to the tabu lists in order to select individuals with dissimilar genotypes and consequently avoid premature convergence to local optima.

Integrative Relay Hybrid GAs

Finally, we undertake the study of *integrative relay hybrid GAs*. In this scheme, usually, a GA is used to perform one or more functions in the master metaheuristic. García-Martínez et al. [41] presented a GA designed specifically to play the role of the simulated annealing neighborhood operator. In particular, a steady-state GA creates one single candidate solution at each iteration, by crossing over the current solution of the master simulated annealing and another one from the population. Afterward, the master simulated annealing applies an acceptance mechanism to decide which solution becomes the new current solution, either the candidate solution or the current one. The other solution is inserted into the population by a replacement strategy. Another approach following this scheme was presented in [75]. In this work, the authors present an iterated local search with a perturbation operator based on a μCHC algorithm. CHC provides high diversity by means of high selective pressure, allowing diverse and promising solutions to be maintained. This behavior is desirable for a GA assuming the work of a perturbation operator.

GA Applications

GAs have had a great measure of success in search and optimization problems. The reason for a great part of their success is their ability to exploit the information accumulated about an initially unknown search space in order to bias subsequent searches into useful subspaces, i.e., their adaptation. This is their key feature, particularly in large, complex, and poorly understood search spaces, where classical search tools (enumerative, heuristic, etc.) are inappropriate, offering a valid approach to problems requiring efficient and effective search techniques.

With the aim of showing the broad variety of GA applications, we have searched for papers from scopus with the keywords "genetic algorithm" in the title (the query was submitted on May 2015). Table 6 outlines the number of documents belonging to different subject areas. The huge amount of papers (43,180) and the wide spectrum of subject areas with GA applications allow us to say that the research on GAs has reached a stage of great maturity, and there is an active and vibrant worldwide community of researchers working on these algorithms, as it may be confirmed in Fig. 3, which illustrates the number of publications appeared in each year in the period 1970–2014. Specifically, we may see that from approximately the last teen years, GAs involved more than 2,500 publications per year (reaching a peak of over 3,500 in 2010).

An additional remark from Table 6 concerns the outstanding activity coming from the engineering field, where challenging hard optimization problems arise, which are characterized by having multiple objectives, by being dynamic problems, by the high number of implied decision variables and the complex relationships among them, and, in many cases, by the strict feasibility constraints. To effectively face problems with such a diverse nature, the original GA scheme was extended in different ways:

- *Multiobjective GAs*. Multiobjective optimization problems require the simultaneous optimization (maximization or minimization) of several objectives that cannot be compared easily with each other. The goal of solving these problems is not to find an optimal solution but rather to find a set of nondominated solutions, the so-called Pareto set. Since the 1980s, the application of GAs in solving multiobjective optimization problems has been receiving a growing interest, and they are still one of the hottest research areas in the field of evolutionary computation [19, 24]. By evolving a population of solutions, these GA approaches are able to capture a set of nondominated solutions in a single run of the algorithm [25].

Table 6 Research papers on GAs by subject areas

Subject area	Documents
Engineering	24,095
Computer science	19,235
Mathematics	7,062
Physics and astronomy	3,058
Materials science	2,140
Energy	1,871
Decision sciences	1,863
Biochemistry, genetics, and molecular biology	1,615
Earth and planetary sciences	1,326
Chemical engineering	1,308
Environmental science	1,267
Social sciences	1,152
Chemistry	1,064
Business, management, and accounting	926
Agricultural and biological sciences	634
Medicine	589
Multidisciplinary	421
Economics, econometrics, and finance	204
Neuroscience	175
Pharmacology, toxicology, and pharmaceutics	140
Immunology and microbiology	100
Health professions	89
Arts and humanities	57
Psychology	25
Undefined	22
Nursing	7
Veterinary	5
Dentistry	2
Total	43,180

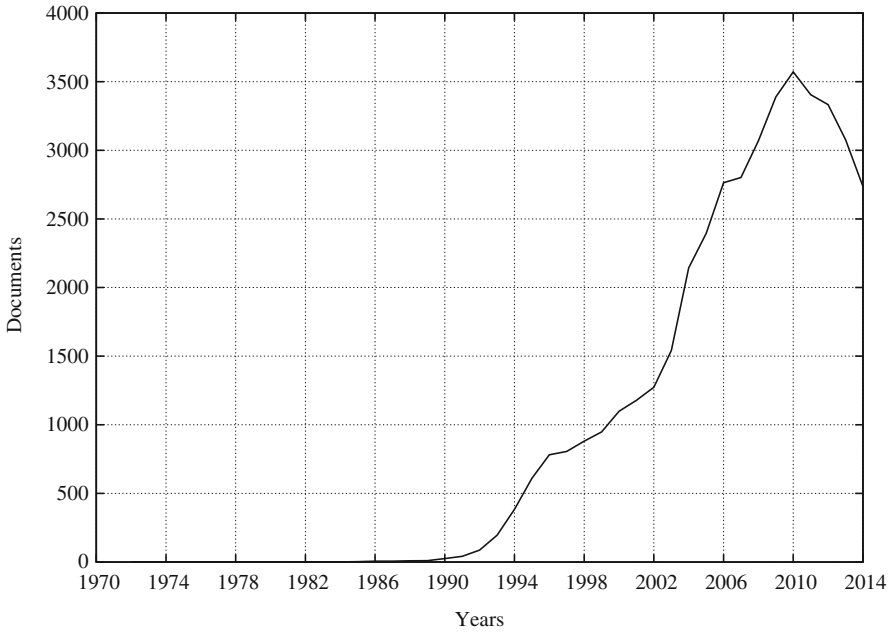

Fig. 3 GA publications per year

- *Constrained GAs*. In many science and engineering disciplines, it is common to encounter a large number of constrained optimization problems. Such problems involve an objective function that is subject to various equality and inequality constraints. The challenges in this optimization scenario arise from the various limits on the decision variables, the constraints involved, the interference among constraints, and the interrelationship between the constraints and the objective functions. Classical gradient-based optimization methods have difficulties in handling this kind of problems, as constrained optimization problems may usually lack an explicit mathematical formulation and have discrete definition domains. GAs are unconstrained search methods and lack of an explicit mechanism to bias the search in constrained search space. However, researchers have been able to tailor constraint handling techniques into GAs [18, 82, 111].
- *GAs for dynamic optimization problems*. In dynamic environments, the fitness landscape may change over time as a result of the changes of the optimization goal, problem instance, and/or some restrictions. Alternatives that were bad in the past can be good nowadays or vice versa; criteria that were important before become irrelevant now, etc., and moving in these dynamic scenarios is a challenge. For these cases, the goal of an optimization algorithm is no longer to find a satisfactory solution to a fixed problem, but to track the moving optimum in the search space as closely as possible. This poses great difficulties to standard GAs, because they cannot adapt well to the changing environment once

converged. Over the past decade, an important number of GA approaches have been developed to face problems with these features, and readers are referred to [21, 62, 116] for a comprehensive overview.

- *Multimodal GAs*. Many real-world problems require an optimization algorithm that is able to explore multiple optima in their search space. However, given a problem with multiple solutions, a simple GA shall tend to converge to a single solution. As a result, various mechanisms have been proposed to stably maintain a diverse population throughout the search [73, 74, 117], thereby allowing GAs to identify multiple optima reliably in these multimodal function landscapes. Many of these methods work by encouraging artificial niche formation through sharing [46] and crowding [52, 97, 106] (see section "Strategies to Maintain Diversity").

Conclusions

GAs have become a tool of choice since they offer practical advantages to researchers facing difficult optimization problems because they may locate high-performance regions of vast and complex search spaces. Other advantages include the simplicity of the approach, their flexibility, and their robust response to changing circumstances. In the previous sections, we firstly provided a comprehensive guide for newcomers, revising the basic ingredients for designing a GA and presenting the classical GA approaches to solve three widely known optimization problems with very different characteristics. Then, we undertook the study of more complex GA instances through two fruitful research lines in this field such as the preservation of diversity in GAs and the hybridizations of GAs with other metaheuristics. With regard to the former, we analyzed the premature convergence problem in GAs and outlined different approaches that have been proposed to introduce new methods and operators that attempt to favor the diversity in GAs. Similarly, we described different schemes in the literature to hybridize GAs with other metaheuristics and presented different instantiations of these schemes that have been successfully applied. Finally, we reviewed the current activity in the GA field in terms of number of publications and areas of interest, which shows the high level of interest in the field of GAs and predicts a promising evolution of this research area.

Cross-References

▸ A History of Metaheuristics
▸ Evolutionary Algorithms
▸ Memetic Algorithms
▸ Multi-objective Optimization
▸ Random-Key Genetic Algorithms

References

1. Adler D (1993) Genetic algorithm and simulated annealing: a marriage proposal. In: Proceedings of the IEEE international conference on neural network, pp 1104–1109
2. Alba E, Dorronsoro B (2005) The exploration/exploitation tradeoff in dynamic cellular genetic algorithms. IEEE Trans Evol Comput 9(2):126–142
3. Alba E, Tomassini M (2002) Parallelism and evolutionary algorithms. IEEE Trans Evol Comput 6(5):443–462
4. Alba E, Troya JM (2001) Analyzing synchronous and asynchronous parallel distributed genetic algorithms. Future Gener Comput Syst 17(4):451–465
5. Al-Naqi A, Erdogan A, Arslan T (2013) Adaptive three-dimensional cellular genetic algorithm for balancing exploration and exploitation processes. Soft Comput 17(7): 1145–1157
6. Araujo L, Merelo J (2011) Diversity through multiculturality: assessing migrant choice policies in an island model. IEEE Trans Evol Comput 15(4):456–469
7. Bäck T, Schütz M (1996) Intelligent mutation rate control in canonical genetic algorithms. In: Raś Z, Michalewicz M (eds) Foundations of intelligent systems. Lecture notes in computer science, vol 1079, pp 158–167
8. Baker J (1987) Adaptive selection methods for genetic algorithms. In: Grefenstette J (ed) International conference on genetic algorithms applications and their application. Erlbaum Associates, pp 14–21
9. Baker J (1987) Reducing bias and inefficiency in the selection algorithm. In: Proceedings of the international conference on genetic algorithms, pp 14–21
10. Baluja S, Caruanna R (1995) Removing the genetics from the standard genetic algorithm. In: Proceedings of the annual conference on machine learning, pp 38–46
11. Banzhaf W (1990) The "Molecular" traveling salesman. Biol Cybern 64:7–14
12. Beyer H, Deb K (2001) On self-adaptive features in real-parameter evolutionary algorithms. IEEE Trans Evol Comput 5(3):250–270
13. Bhandarkar S, Zhang H (1999) Image segmentation using evolutionary computation. IEEE Trans Evol Comput 3(1):1–21
14. Blum C (2010) Hybrid metaheuristics – guest editorial. Comput Oper Res 37(3):430–431
15. Blum C, Puchinger J, Raidl G, Roli A (2011) Hybrid metaheuristics in combinatorial optimization: a survey. Appl Soft Comput 11:4135–4151
16. Caruana R, Schaffer J (1988) Representation and hidden bias: gray versus binary coding for genetic algorithms. In: Proceedings of the fifth international conference on machine learning, pp 153–162
17. Chelouah R, Siarry P (2003) Genetic and Nelder-Mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions. Eur J Oper Res 148(2): 335–348
18. Coello CAC (2002) Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Comput Methods Appl Mech Eng 191(11–12):1245–1287
19. Coello C, Lamont G, Veldhuizen D (2006) Evolutionary algorithms for solving multi-objective problems. Springer, New York
20. Črepinšek M, Liu SH, Mernik M (2013) Exploration and exploitation in evolutionary algorithms: a survey. ACM Comput Surv 45(3):35:1–35:33
21. Cruz C, González JR, Pelta DA (2011) Optimization in dynamic environments: a survey on problems, methods and measures. Soft Comput 15(7):1427–1448
22. Dantzig G (1957) Discrete variable extremum problems. Oper Res 5:266–277
23. Davis L (1985) Adaptive algorithms to epistactic domains. In: Proceedings of the international conference on artificial intelligence, pp 162–164
24. Deb K (2001) Multi-objective optimization using evolutionary algorithms. Wiley, Chichester/New York

25. Deb K (2008) Introduction to evolutionary multiobjective optimization. In: Branke J, Deb K, Miettinen K, Słowiski R (eds) Multiobjective optimization. Lecture notes in computer science, vol 5252. Springer, Berlin/Heidelberg, pp 59–96
26. De Jong K (1975) An analysis of the behavior of a class of genetic adaptive systems. PhD thesis, University of Michigan
27. De Jong K (1993) Genetic algorithms are NOT function optimizers. In: Whitley LD (ed) Foundations of genetic algorithms 2. Morgan Kaufmann, San Mateo
28. De Jong K, Sarma J (1993) Generation gaps revisited. In: Whitley LD (ed) Foundations of genetic algorithms. Morgan Kaufmann, San Mateo, pp 19–28
29. Eiben A, Smith J (2003) Introduction to evolutionary computation. Natural computing series. Springer, New York
30. Eiben A, Hinterding R, Michalewicz Z (1999) Parameter control in evolutionary algorithms. IEEE Trans Evol Comput 3(2):124–141
31. Eklund SE (2004) A massively parallel architecture for distributed genetic algorithms. Parallel Comput 30(5–6):647–676
32. Eshelman L (1991) The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination. Foundations of genetic algorithms, vol 1. Morgan Kaufmann, San Mateo, CA, pp 265–283
33. Eshelman L, Schaffer J (1991) Preventing premature convergence in genetic algorithms be preventing incest. In: Proceedings of the international conference on genetic algorithms, pp 115–122
34. Eshelman L, Schaffer J (1993) Real-coded genetic algorithms and interval-schemata. In: Whitley LD (ed) Foundations of genetic algorithms 2. Morgan Kaufmann, San Mateo, pp 187–202
35. Fernandes C, Rosa A (2008) Self-adjusting the intensity of assortative mating in genetic algorithms. Soft Comput 12(10):955–979
36. Fogarty TC (1989) Varying the probability of mutation in the genetic algorithm. In: Proceedings of the third international conference on genetic algorithms, pp 104–109
37. Fogel D (1998) Evolutionary computation: the fossil record. IEEE Press, New York
38. Gao J, Sun L, Gen M (2008) A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Comput Oper Res 35(9): 2892–2907
39. García-Martínez C, Lozano M, Herrera F, Molina D, Sánchez A (2008) Global and local real-coded genetic algorithms based on parent-centric crossover operators. Eur J Oper Res 185:1088–1113
40. García-Martínez C, Lozano M, Herrera F, Molina D, Sánchez A (2008) Global and local real-coded genetic algorithms based on parent-centric crossover operators. Eur J Oper Res 185(3):1088–1113
41. García-Martínez C, Lozano M, Rodríguez-Díaz F (2012) A simulated annealing method based on a specialised evolutionary algorithm. Appl Soft Comput 12(2):573–588
42. Ghannadian F, Alford C, Shonkwiler R (1996) Application of random restart to genetic algorithms. Inf Sci 95(1–2):81–102
43. Goldberg D (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, New York
44. Goldberg D (1989) Sizing populations for serial and parallel genetic algorithms. In: Schaffer J (ed) International conference on genetic algorithms. Morgan Kaufmann, pp 70–79
45. Goldberg D, Lingle R (1985) Alleles, Loci and the traveling salesman problem. In: Proceedings of the international conference on genetic algorithms, pp 154–159
46. Goldberg D, Richardson J (1987) Genetic algorithms with sharing for multimodal function optimization. In: Grefenstette J (ed) Proceedings of the international conference on genetic algorithms. L. Erlbraum Associates, pp 41–49
47. Goldberg D, Korb B, Deb K (1990) Messy genetic algorithms: motivation, analysis, and first results. Complex Syst 3:493–530

48. Gonçalves JF, Resende MG (2011) Biased random-key genetic algorithms for combinatorial optimization. J Heuristics 17(5):487–525
49. Grosan C, Abraham A (2007) Hybrid evolutionary algorithms: methodologies, architectures, and reviews. In: Grosan C, Abraham A, Ishibuchi H (eds) Hybrid evolutionary algorithms. Springer, Berlin/New York, pp 1–17
50. Grötschel M, Padberg MM (1978) On the symmetric traveling salesman problem: theory and computations. In: Optimization and operations research. Lecture notes in econocmics and mathematical systems, vol 157. Springer, pp 105–115
51. Gupta S, Garg ML (2013) Binary trie coding scheme: an intelligent genetic algorithm avoiding premature convergence. Int J Comput Math 90(5):881–902
52. Harik G (1995) Finding multimodal solutions using restricted tournament selection. In: Proceedings of the international conference on genetic algorithms. Morgan Kaufmann, pp 24–31
53. Herrera F, Lozano M (2000) Gradual distributed real-coded genetic algorithms. IEEE Trans Evol Comput 4(1):43–63
54. Herrera F, Lozano M (2003) Fuzzy adaptive genetic algorithms: design, taxonomy, and future directions. Soft Comput 7(8):545–562
55. Herrera F, Lozano M, Verdegay J (1998) Tackling real-coded genetic algorithms: operators and tools for behavioural analysis. Artif Intell Rev 12:265–319
56. Herrera F, Lozano M, Sánchez A (2003) A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study. Int J Intell Syst 18(3):309–338
57. Hinterding R, Michalewicz Z, Eiben A (1997) Adaptation in evolutionary computation: a survey. In: IEEE international conference on evolutionary computation, pp 65–69
58. Holland J (1975) Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor
59. Hutter M, Legg S (2006) Fitness uniform optimization. IEEE Trans Evol Comput 10(5): 568–589
60. Iman R, Conover W (1982) A distribution-free approach to inducing rank correlation among input variables. Commun Stat Simul Comput 11(3):311–334
61. Janikow C, Michalewicz Z (1991) An experimental comparison of binary and floating point representation in genetic algorithms. In: Proceedings of the fourth international conference on genetic algorithms, pp 31–36
62. Jin Y, Branke J (2005) Evolutionary optimization in uncertain environments-a survey. IEEE Trans Evol Comput 9(3):303–317
63. Karafotias G, Hoogendoorn M, Eiben A (2015) Parameter control in evolutionary algorithms: trends and challenges. IEEE Trans Evol Comput 19(2):167–187
64. Kazarlis S, Papadakis S, Theocharis J, Petridis V (2001) Microgenetic algorithms as generalized hill-climbing operators for GA optimization. IEEE Trans Evol Comput 5(3):204–217
65. Kazimipour B, Li X, Qin A (2014) A review of population initialization techniques for evolutionary algorithms. In: Proceedings of the IEEE congress on evolutionary computation, pp 2585–2592
66. Kita H (2001) A comparison study of self-adaptation in evolution strategies and real-coded genetic algorithms. Evol Comput 9(2):223–241
67. Kominami M, Hamagami T (2007) A new genetic algorithm with diploid chromosomes by using probability decoding for non-stationary function optimization. In: IEEE international conference on systems, man and cybernetics, 2007. ISIC. pp 1268–1273
68. Koumousis V, Katsaras C (2006) A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. IEEE Trans Evol Comput 10(1):19–28
69. Krishnakumar K (1989) Micro-genetic algorithms for stationary and non-stationary function optimization. In: Intelligent control and adaptive systems. Proceedings of the SPIE, vol 1196, pp 289–296

70. Kuo T, Hwang S (1996) A genetic algorithm with disruptive selection. IEEE Trans Syst Man Cybern 26(2):299–307
71. Kurahashi S, Terano T (2000) A genetic algorithm with tabu search for multimodal and multiobjective function optimization. In: Whitley LD, Goldberg DE, Cant-Paz E, Spector L, Parmee IC, Beyer HG (eds) GECCO. Morgan Kaufmann, pp 291–298
72. Larrañaga P, Kuijpers C, Murga R, Inza I, Dizdarevic S (1999) Genetic algorithms for the travelling salesman problem: a review of representations and operators. Artif Intell Rev 13(2):129–170
73. Li JP, Balazs ME, Parks GT, Clarkson PJ (2002) A species conserving genetic algorithm for multimodal function optimization. Evol Comput 10(3):207–234
74. Liang Y, Leung KS (2011) Genetic algorithm with adaptive elitist-population strategies for multimodal function optimization. Appl Soft Comput 11(2):2017–2034
75. Lozano M, García-Martínez C (2010) Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: overview and progress report. Comput Oper Res 37:481–497
76. Lozano M, Herrera F, Krasnogor N, Molina D (2004) Real-coded memetic algorithms with crossover hill-climbing. Evol Comput 12(3):273–302
77. Lozano M, Herrera F, Cano JR (2008) Replacement strategies to preserve useful diversity in steady-state genetic algorithms. Inf Sci 178(23):4421–4433
78. Mahfoud S (1992) Crowding and preselection revised. In: Männer R, Manderick B (eds) Parallel problem solving from nature, vol 2. Elsevier Science, pp 27–36
79. Mallipeddi R, Suganthan P (2010) Ensemble of constraint handling techniques. IEEE Trans Evol Comput 14(4):561–579
80. Mauldin M (1984) Maintaining diversity in genetic search. In: National conference on artificial intelligence, Austin, pp 247–250
81. Michalewicz Z (1992) Genetic algorithms + data structures = evolution programs. Springer, Berlin/New York
82. Michalewicz A, Schoenauer M (1996) Evolutionary algorithms for constrained parameter optimization problems. Evol Comput 4(1):1–32
83. Molina D, Lozano M, García-Martínez C, Herrera F (2010) Memetic algorithms for continuous optimization based on local search chains. Evol Comput 18(1):27–63
84. Moscato P, Cotta C (2003) A gentle introduction to memetic algorithms. In: Glover F, Kochenberger GA (eds) Handbook of metaheuristics. Kluwer Academic, Boston, pp 105–144
85. Noman N, Iba H (2008) Accelerating differential evolution using an adaptive local search. IEEE Trans Evol Comput 12(1):107–125
86. Nomura T, Shimohara K (2001) An analysis of two-parent recombinations for real-valued chromosomes in an infinite population. Evol Comput 9(3):283–308
87. Oh IS, Lee JS, Moon BR (2004) Hybrid genetic algorithms for feature selection. IEEE Trans Pattern Anal Mach Intell 26(11):1424–1437
88. Oliver I, Smith D, Holland J (1987) A study of permutation crossover operators on the TSP. In: Proceedings of the international conference on genetic algorithms and their applications, pp 224–230
89. Pereira A, de Andrade BB (2015) On the genetic algorithm with adaptive mutation rate and selected statistical applications. Comput Stat 30(1):131–150
90. Potts J, Giddens T, Yadav S (1994) The development and evaluation of an improved genetic algorithm based on migration and artificial selection. IEEE Trans Syst Man Cybern 24: 73–86
91. Preechakul C, Kheawhom S (2009) Modified genetic algorithm with sampling techniques for chemical engineering optimization. J Ind Eng Chem 15:110–118
92. Preux P, Talbi E (1999) Towards hybrid evolutionary algorithms. Int Trans Oper Res 6(6): 557–570

93. Raidl G (2006) A unified view on hybrid metaheuristics. In: Almeida F, Aguilera MB, Blum C, Vega JM, Pérez MP, Roli A, Sampels M (eds) Hybrid metaheuristics, LNCS, vol 4030. Springer, pp 1–12
94. Reeves C (2010) Genetic algorithms. In: Gendreau M, Potvin J-Y (eds) Handbook of metaheuristics, vol 146. Springer, New York, pp 109–139
95. Reeves C, Rowe J (2001) Genetic algorithms: principles and perspectives. Kluwer, Norwell
96. Rodriguez F, Garcia-Martinez C, Lozano M (2012) Hybrid metaheuristics based on evolutionary algorithms and simulated annealing: taxonomy, comparison, and synergy test. IEEE Trans Evol Comput 16(6):787–800
97. Sareni B, Krahenbuhl L (1998) Fitness sharing and Niching methods revisited. IEEE Trans Evol Comput 2(3):97–106
98. Serpell M, Smith J (2010) Self-adaption of mutation operator and probability for permutation representations in genetic algorithms. Evol Comput 18(3):491–514
99. Smith JE, Fogarty TC (1997) Operator and parameter adaptation in genetic algorithms. Soft Comput 1(2):81–87
100. Smith A, Coit D, Baeck T, Fogel D, Michalewicz Z (1997) Penalty functions. In: Bäck T, Fogel DB, Michalewics Z (eds) Handbook on evolutionary computation. Oxford University Press, New York, pp C5.2:1–C5.2:6
101. Srinivas M, Patnaik L (1994) Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Trans Syst Man Cybern 24(4):656–667
102. Syswerda G (1989) Uniform crossover in genetic algorithms. In: Proceedings of the international conference on genetic algorithms, pp 2–9
103. Talbi E (2002) A taxonomy of hybrid metaheuristics. J Heuristics 8(5):541–564
104. Talbi EG, Bachelet V (2006) Cosearch: a parallel cooperative metaheuristic. J Math Model Algorithms 5(1):5–22
105. Tantar A, Melab N, Talbi E (2008) A grid-based genetic algorithm combined with an adaptive simulated annealing for protein structure prediction. Soft Comput 12(12):1185–1198
106. Thierens D (1998) Selection schemes, elitist recombination, and selection intensity. In: Proceedings of the 7th international conference on genetic algorithms. Morgan Kaufmann, pp 152–159
107. Ting CK, Li ST, Lee C (2003) On the harmonious mating strategy through tabu search. Inf Sci 156:189–214
108. Tuson A, Ross P (1998) Adapting operator settings in genetic algorithms. Evol Comput 6(2):161–184
109. Uyar Ai, Harmanci AE (2005) A new population based adaptive domination change mechanism for diploid genetic algorithms in dynamic environments. Soft Comput 9(11): 803–814
110. van Kemenade C, Kok J, Eiben AE (1995) Raising GA performance by simultaneous tuning of selective pressure and recombination disruptiveness. In: Proceedings of the 1995 IEEE congress on evolutionary computation (CEC 1995), pp 346–351
111. Venkatraman S, Yen G (2005) A generic framework for constrained optimization using genetic algorithms. IEEE Trans Evol Comput 9(4):424–435
112. Vrugt J, Robinson B, Hyman J (2009) Self-adaptive multimethod search for global optimization in real-parameter spaces. IEEE Trans Evol Comput 13(2):243–259
113. Whitley D (1989) The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. In: Proceedings of the international conference on genetic algorithms. Morgan Kaufmann, pp 116–121
114. Wong YY, Lee KH, Leung KS, Ho CW (2003) A novel approach in parameter adaptation and diversity maintenance for genetic algorithms. Soft Comput 7(8):506–515
115. Yang CH, Nygard K (1993) Effects of initial population in genetic search for time constrained traveling salesman problems. In: Proceedings of the ACM computer science conference, pp 378–383

116. Yang S, Ong Y, Jin Y (eds) (2007) Evolutionary computation in dynamic and uncertain environments. Studies in computational intelligence, vol 51. Springer, Berlin/London
117. Yao J, Kharma N, Grogono P (2010) Bi-objective multipopulation genetic algorithm for multimodal function optimization. IEEE Trans Evol Comput 14(1):80–102
118. Yeniay Ö (2005) Penalty function methods for constrained optimization with genetic algorithms. Math Comput Appl 10(1):45–56
119. Yuen SY, Chow CK (2009) A genetic algorithm that adaptively mutates and never revisits. IEEE Trans Evol Comput 13(2):454–472