Genetic Algorithms

Martin Pelikan

MEDAL Report No. 2010007

August 2010

Abstract

Genetic algorithms [1, 2] are stochastic optimization methods inspired by natural evolution and genetics. Over the last few decades, genetic algorithms have been successfully applied to many problems of business, engineering, and science. Because of their operational simplicity and wide applicability, genetic algorithms are now playing an increasingly important role in computational optimization and operations research. This article provides an introduction to genetic algorithms as well as numerous pointers for obtaining additional information.

Keywords

Genetic algorithms, evolutionary computation.

Missouri Estimation of Distribution Algorithms Laboratory (MEDAL)
Department of Mathematics and Computer Science
University of Missouri–St. Louis
One University Blvd., St. Louis, MO 63121
E-mail: medal@cs.umsl.edu
WWW: http://medal.cs.umsl.edu/

Genetic Algorithms

Martin Pelikan
Missouri Estimation of Distribution Algorithms Laboratory (MEDAL)
Dept. of Math and Computer Science, 320 CCB
University of Missouri at St. Louis
One University Blvd., St. Louis, MO 63121
`pelikan@cs.umsl.edu`

August 28, 2010

Abstract

Genetic algorithms [1, 2] are stochastic optimization methods inspired by natural evolution and genetics. Over the last few decades, genetic algorithms have been successfully applied to many problems of business, engineering, and science. Because of their operational simplicity and wide applicability, genetic algorithms are now playing an increasingly important role in computational optimization and operations research. This article provides an introduction to genetic algorithms as well as numerous pointers for obtaining additional information.

1 Introduction

Genetic algorithms (GAs) [1, 2] work with a population or multiset of candidate solutions. Maintaining a *population of candidate solutions*—as opposed to a *single solution*—has several advantages. For example, using a population allows simultaneous exploration of multiple basins of attraction, it allows statistical decision making based on the entire sample of promising solutions even when the evaluation procedure is affected by external noise, and it enables the use of learning techniques to identify problem regularities. The initial population is typically generated at random using the uniform distribution over all admissible solutions. The population is then updated for a number of iterations using the operators of selection, variation, and replacement. The selection operator ensures that the search for the optimum focuses on candidate solutions of high quality. Variation operators exploit information from the best solutions found so far to generate new high quality solutions. Finally, the replacement operator provides a method to update the original population of solutions using the new solutions created by selection and variation.

The purpose of this article is to provide an introduction to GAs. The article is organized as follows. Section 2 outlines the basic GA procedure. Section 3 details popular GA operators and discusses how GAs can be applied to problems defined over various representations. Section 4 focuses on the use of GAs for solving problems with multiple objectives, noise and constraints. Section 5 outlines several approaches to enhancing efficiency of GAs. Section 6 provides basic guidelines for applying GAs to a new problem. Finally, section 7 outlines some of the main starting points for the reader interested in learning more about GAs.

2 Genetic Algorithm Procedure

An optimization problem may be defined by specifying (1) a representation of potential solutions to the problem and (2) a measure for evaluating quality of each candidate solution. The goal is to find a solution or a set of solutions that perform best with respect to the specified measure. For example, in the traveling salesman problem, potential solutions are represented by permutations of cities to visit and the measure of solution quality gives preference to shorter tours.

GAs pose no hard restrictions on the representation of candidate solutions or the measure for evaluating solution quality, although it is important that the underlying representation is consistent with the chosen operators. Representations can vary from binary strings to vectors of real numbers, to permutations, to production rules, to schedules, and to program codes. Performance measures can be based on a computer procedure, a simulation, an interaction with the human, or a combination of the above. Solution quality can also be measured using a partial ordering operator on the space of candidate solutions, which only compares relative quality of two solutions without expressing solution quality numerically. Additionally, evaluation can be probabilistic and it can be affected by external noise. Nonetheless, for the sake of simplicity, in this section it is assumed that candidate solutions are represented by binary strings of fixed length and that the performance of each candidate solution is represented by a real number called *fitness*. The task is to find a binary string or a set of binary strings with the highest fitness.

2.1 Components of a Genetic Algorithm

The basic GA procedure consists of the following key ingredients:

Initialization. GAs usually generate the initial population of candidate solutions randomly according to a uniform distribution over all admissible solutions. However, the initial population can sometimes be biased using prior problem-specific knowledge or other optimization procedures [3, 4, 5, 6].

Selection. Each GA iteration starts by selecting a set of promising solutions from the current population based on the quality of each solution. A number of different selection techniques can be used [7, 8], but the basic idea of all these methods is the same—make more copies of solutions that perform better at the expense of solutions that perform worse. For example, *tournament selection* of order s selects one solution at a time by first choosing a random subset of s candidate solutions from the current population and then selecting the best solution out of this subset [9]. Random tournaments are repeated until there are sufficiently many solutions in the selected population. The size of the tournaments determines selection pressure—the larger the tournaments, the higher the pressure on the quality of each solution. The tournament size of $s = 2$ is a common setting. Typically, the size of the selected set of solutions is the same as the size of the original population.

Variation. Once the set of promising solutions has been selected, new candidate solutions are created by applying recombination (crossover) and mutation to the promising solutions. Crossover combines subsets of promising solutions by exchanging some of their parts. Mutation perturbs the recombined solutions slightly to explore their immediate neighborhood. Most of the commonly used crossover operators combine pairs of promising solutions. For example, *one-point crossover* randomly selects a single position in the two strings (crossing site) and exchanges the bits on all the subsequent positions (see Figure 1). Typically, the selected population is first randomly divided into pairs of candidate solutions, and crossover

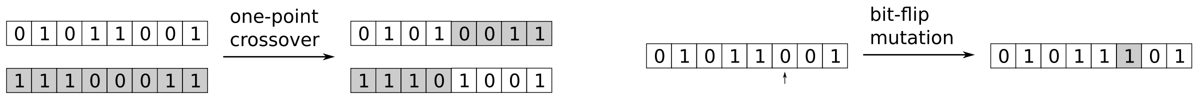


Figure 1: In one-point crossover, a site is randomly chosen and the strings exchange all bits after this site. In bit-flip mutation, each bit is modified with the same probability. The probability is typically small and only one or a few bits are changed.

is then applied to each of these pairs with a specified probability p_c. The probability p_c of crossover is usually in the range $[0.6, 0.95]$. One of the most common mutation operators for binary strings is *bit-flip mutation*, in which each bit is modified (flipped) with the same probability (see Figure 1); the probability of changing a bit is typically small, changing only one or a few bits at a time.

Replacement. After applying crossover and mutation to the set of promising solutions, the population of new candidate solutions replaces the original one or its part, and the next iteration is executed (starting with selection) unless termination criteria are met. For example, the run can be terminated when the population converges to a singleton, the population contains a good enough solution, or an upper bound on the number of iterations has been reached.

The pseudocode of the basic genetic algorithm follows:

```
Genetic algorithm {
   t = 0;
   generate initial population P(0);
   while (not done) {
      select population of promising solutions S(t) from P(t);
      apply variation operators on S(t) to create O(t);
      create P(t+1) by combining O(t) and P(t);
      t = t+1;
   }
}
```

Since genetic algorithms are inspired by biology, common GA terminology is strongly influenced by that in biology; see Table 1 for an overview of GA terminology.

Table 1: Common GA terminology.

Term	**Alternative name(s)**
candidate solution	individual, chromosome, string
decision variable	variable, locus, string position
value of decision variable	bit, allele
measure of solution quality	fitness function, objective function
numeric representation of solution quality	fitness, fitness value
population of promising solutions, $S(t)$	parent population, parents, selected solutions
population of new solutions, $O(t)$	offspring population, offspring, children
iteration	generation

2.2 Simulation of a Genetic Algorithm by Hand

To illustrate the basic GA procedure, consider solving the onemax problem, which assumes that candidate solutions are represented by binary strings of n bits and the fitness is computed as the sum of bits in the input binary string:

$$onemax(X_1, X_2, \ldots, X_n) = \sum_{i=1}^{n} X_i,$$

where $(X_1, X_2, \ldots, X_n)$ denotes the input binary string. Onemax is a simple unimodal function with the optimum in the string $(1, 1, \ldots, 1)$.

The simulation will use binary tournament selection (i.e., s=2), one-point crossover, and full replacement (the new population of solutions fully replaces the original population). The remaining parameters of the simulation by hand will be set as follows:

parameter	description	value
n	string length	8
N	population size	10
p_c	crossover probability	0.6
p_m	mutation probability	1/8

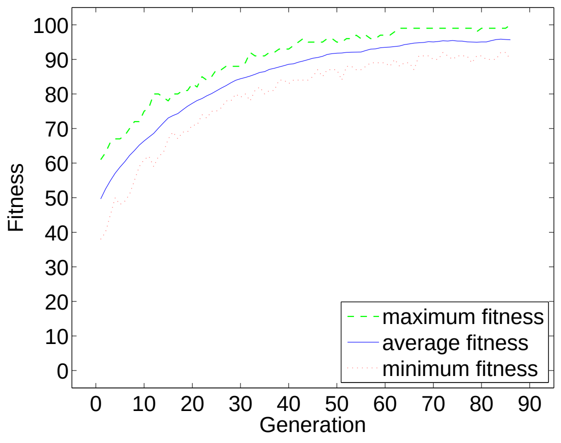
The first step consists of generating $N = 10$ binary strings of length $n = 8$. The resulting initial population is shown below:

	candidate solution	fitness		candidate solution	fitness
1	10111001	5	6	00001010	2
2	00010010	2	7	11011000	4
3	01101001	4	8	11111000	5
4	00001001	2	9	11111000	5
5	00100110	3	10	00011001	3

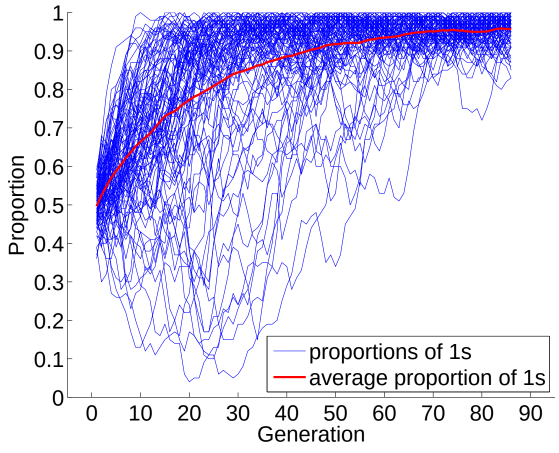
Selection of the parent population starts by random selection of 10 pairs of candidate solutions from the original population. The winner of each of these tournaments is selected based on the fitness function (for ties the winner is chosen arbitrarily):

	tournament	winner	fitness		tournament	winner	fitness
1	11111000 & 00011001	11111000	5	6	00010010 & 11111000	11111000	5
2	00010010 & 00011001	00011001	3	7	01101001 & 10111001	10111001	5
3	11011000 & 10111001	10111001	5	8	00001001 & 00010010	00010010	2
4	01101001 & 00001010	01101001	4	9	00100110 & 11011000	11011000	4
5	00100110 & 11111000	11111000	5	10	10111001 & 00011001	10111001	5

Since each parent solution was chosen independently, it is not necessary to pair solutions for crossover randomly but consequent pairs of selected strings can be considered for crossover. There are 5 pairs of parents and the probability of crossover is 0.6; therefore, about 3 pairs of parents are expected to be recombined using crossover. Mutation is then applied to all the resulting strings. Mutation is expected to change 1 bit on average since $p_m = 1/8$ and $n = 8$. The results after applying crossover and mutation follow:



(a) Fitness values.



(b) Proportions of 1s in different positions.

Figure 2: GA simulation on onemax of $n = 100$ bits.

	parents	crossing site	after crossover	after mutation	fitness
1	11111000	3	$11\underline{011001}$	$11\underline{1}11001$	6
2	00011001		$00\underline{111000}$	00111000	3
3	10111001	-	10111001	$1\underline{1}111001$	6
4	01101001		01101001	$0\underline{0}1\underline{1}1\underline{1}01$	5
5	11111000	2	$1\underline{1111000}$	$1111\underline{0}000$	4
6	11111000		$1\underline{1111000}$	11111000	5
7	10111001	-	10111001	$10\underline{0}110\underline{1}1$	5
8	00010010		00010010	$\underline{1}0010010$	3
9	11011000	6	$11011\underline{001}$	$1101\underline{0}001$	4
10	10111001		$10111\underline{000}$	10111000	4

The population after mutation will then replace the original population of solutions, which serves as the starting point for the next GA iteration.

One of the ways to analyze the results of the simulation is to look at the change in the average fitness over the population. The average fitness of the original population is 3.5, whereas the average fitness of the new population is 4.5. Therefore, using selection, crossover, and mutation increased the average quality of solutions in the population. At the same time, the new solutions are different than those in the original population, although they share many similarities. We were thus able to create a population of new solutions, which are of better quality than those that we started with. Simulating the following few iterations of the GA should further increase solution quality, until the global optimum `11111111` would be found.

2.3 Simulation of a Genetic Algorithm on a Larger Problem

In this section, the simulation of the GA by hand is extended to deal with a larger onemax problem of $n = 100$ bits. Due to the increased problem size, in order to ensure that the global optimum will be found with a high probability, we also increase the population size. Specifically, we set the population size as $N = 100$, the probability of crossover as $p_c = 0.6$, and the probability of mutation as $p_m = 1/100$. The results of the simulation are presented in Figure 2.

The results presented in Figure 2(a) show that the average population fitness increases over time and so does the maximum and minimum fitness in the population, although the minimum and maximum fitness often decrease temporarily. Furthermore, Figure 2(b) shows that the proportions of 1s in most string positions increase over time, although some of these proportions decrease temporarily and fluctuate substantially. Nonetheless, the average proportion of 1s in different positions steadily increases and eventually the probability of a 1 in any position becomes 80% or more. Continuing the simulation would lead to a further increase in the proportions of 1s until almost every solution in the population would be globally optimal.

Of course, onemax is a relatively simple problem and it is not a surprise that GAs can solve this problem without any difficulties. Onemax is used to only illustrate the basic GA procedure on a problem that is easy to understand. For onemax, theoretical models [10, 11, 12] exist which estimate that, assuming that the expected number of bits that do not converge to the correct values is upper bounded by an arbitrary constant, the number of evaluations until convergence is expected to grow at most as $O(n \log n)$ where n is the number of bits in the problem.

3 Genetic Algorithm Operators

Previous section presented the basic GA procedure and some of the simplest GA operators. This section presents several other popular selection, replacement and variation operators. Additionally, the section discusses how GAs can be used to solve problems where candidate solutions are not represented by fixed-length binary strings and how linkage learning can be incorporated into GAs to improve their performance and scalability.

3.1 Selection

The main task of the selection operator is to select a population of promising candidate solutions from the current population of candidates, ensuring that solutions of higher quality have a higher probability of being selected than others. There are two main types of selection methods: (1) Fitness proportionate selection and (2) ordinal selection.

Fitness proportionate selection. In the *fitness proportionate selection* methods, each selected solution is drawn from the same probability distribution and the probability of selecting the solution is in some way proportional to its fitness value. Most commonly, the fitness is assumed to be positive and the probability of selecting an individual X with fitness $f(X)$ is linearly proportional to its fitness. Therefore, the probability $p_{select}(X)$ of selecting X from population $P(t)$ is

$$p_{select}(X) = \frac{f(X)}{\sum_{Y \in P(t)} f(Y)}.$$

Ordinal selection. In *ordinal selection* methods, the probability of selecting a particular member of the population does not directly depend on the actual value of its fitness but it depends on the relative quality of this solution compared other members of the current population. *Tournament selection* discussed in the previous section is one of the most popular ordinal selection methods [9]. Another popular ordinal selection method is *truncation selection*, which selects a given percentage τ of the best solutions from the population [11]. For example, for $\tau = 0.5$, the best half of the population is selected. If the selected population should have the same size as the original one, multiple copies of each candidate solution in the selected portion of the population can be created.

Ordinal selection methods are invariant to linear transformations of fitness and they pose fewer restrictions on the fitness function than fitness proportionate selection methods do. Additionally, ordinal selection methods enable a sustained pressure toward solutions of higher quality and the strength of this pressure is often easier to control. These are the main reasons why ordinal selection methods are generally more popular than fitness proportionate selection methods.

The selection operator can also be used to promote population diversity by adjusting the probability of selecting each candidate solution based on how this solution differs from the solutions that have already been selected. To promote diversity, preference should be given to solutions that differ from the solutions selected so far. Methods that aim at maintaining diversity of GA populations are often referred to as *niching methods* [13, 14, 15, 16].

One of the most popular approaches to niching via selection is *fitness sharing*, in which fitness values are updated so that the individuals similar to the previously selected individuals are penalized [17, 15, 16, 18, 13].

3.2 Replacement

The main task of the replacement operator is to incorporate the population of new candidate solutions into the original population of candidates. There are two main approaches to replacement: (1) delete-all and (2) steady-state.

Delete-all or full replacement. With delete-all or full replacement, the entire original population is replaced with the new solutions as was done in the simulations presented in the previous section.

Steady-state replacement. It may often be beneficial to allow candidate solutions in the original population to become a part of the new population, providing overlap between the populations in subsequent iterations of the GA. One way to accomplish this is to combine the original population of candidate solutions with the newly generated ones, and then select the best solutions out of the combined population to form the population for the next iteration. Another approach is to use selection and variation to create fewer individuals than is the size of the original population, and then replace only some of the original solutions with the new ones. The solutions to replace may be selected at random or they may be chosen to be the worst solutions in the original population. GAs where the best solutions found so far are guaranteed to survive to the next generation are often referred to as *elitist* GAs. The proportion of the population that is going to be replaced by new individuals is typically called *generation gap*. GAs with small generation gap in which a substantial portion of the original population survives to the next generation are often called *steady state* GAs. In practice, steady-state replacement schemes are more popular than delete-all replacement.

Replacement can also be used to promote diversity in the population and ensure that the populations of candidate solutions represent a diverse sample of candidate solutions. For example, in a crowding factor model (or, more simply, crowding) [19], for each new individual, a subset of CF candidate solutions is first selected from the original population. The new solution then replaces the most similar solution from this subset. The parameter CF is called crowding factor. The crowding factor provides a way to control crowding; the higher the value of CF, the more pressure on diversity maintenance. Another replacement strategy that promotes diversity is called restricted tournament selection [20]. Restricted tournament selection proceeds similarly to crowding; however, the most similar solution from the selected subset is replaced only if the new solution is better (with respect to the objective).

3.3 Variation

One-point crossover and bit-flip mutation are among the simplest and most popular variation operators. However, many other variation operators are common in practice and this subsection reviews a few such operators applicable to candidate solutions represented by binary strings with the focus on crossover operators applicable to two parents. The topic of variation will be revisited in subsections 3.4 and 3.5, which will discuss linkage learning and some of the more advanced variation operators.

One-point, two-point and k-point crossover. In *one-point crossover*, a crossing site is chosen at random and the two parents exchange all bits after the selected position. On the other hand, in *two-point crossover*, two crossing sites are randomly selected and the two parents exchange all bits between these two sites. The two-point crossover can be further generalized to k-point crossover, in which k crossing sites are selected. Two-point crossover can be approximated by two applications of one-point crossover, whereas k-point crossover can be approximated by k applications of one-point crossover. Most frequently, one-point or two-point crossover operators are used.

Uniform crossover. *Uniform crossover* exchanges the bits in each position of the two parents with probability 50%. About half of the bits are expected to be exchanged on average.

The main difference between k-point crossover and uniform crossover lies in the way these operators select the positions in which the bits are going to be exchanged. In one-point crossover or, more generally, in k-point crossover there is an implicit linkage between positions located close to each other in solution strings. This is because positions that are close to each other are likely to be treated together (either exchanged or not). On the other hand, uniform crossover treats every string position independently, and it is therefore independent of the ordering of decision variables in the solution strings.

It is of note that in GAs crossover is often considered the primary variation operator [2, 21] whereas in evolution strategies [22, 23] mutation is considered the primary variation operator. Consequently, the probability of crossover in GAs is typically quite high (60% or more) and the strength of mutation is set so that mutation causes only small changes in the solution. For an introduction to evolution strategies, see for example refs. [24, 25].

3.4 Dealing with Other Representations

In practice, candidate solutions are often not represented as binary strings of fixed length; for example, candidate solutions may be represented by permutations, vectors of real values, or variable length strings. There are two main approaches to applying GAs to such problems:

Map binary strings to the original representation. Introduce a mapping from binary strings of fixed length to candidate solutions in the original representation. Then, a standard GA for binary strings can be applied and the mapping function can be used to map each obtained candidate solution to the original representation so that the solution can be evaluated. In some cases, it may be difficult to design a one-to-one mapping and, as a result, not all admissible solutions will be considered. Another potential complication is that it may sometimes be far from straightforward to design an appropriate mapping between the set of binary strings and the actual problem representation. On the other hand, the mapping may introduce additional opportunities for discovery and exploitation of problem regularities, which may not be apparent in the original representation.

Modify variation operators. Modify variation operators to make them work with the other representation. Extensions of standard variation operators to some representations are straightforward. For example, if candidate solutions are represented by strings over a finite (non-binary) alphabet, all aforementioned crossover operators can be applied without any modification and the bit-flip mutation can be adjusted to change the value being modified to any other admissible value. Nonetheless, if the used representation is qualitatively different, variation operators may have to be modified.

The remainder of this section discusses the two approaches to dealing with other representations using GAs through examples for two of the most popular representations: (1) real-valued vectors and (2) permutations.

3.4.1 Mapping Between Other Representations and Binary Strings

Solutions to many real-world problems can be encoded as vectors of real-valued variables. There are numerous ways for mapping real-valued vectors into binary strings and the other way around; however, since each real-valued variable can obtain infinitely many values, the mapping cannot be one-to-one and some accuracy will have to be lost when representing real-valued vectors using binary strings of fixed length. For simplicity, let us assume that each real-valued variable can obtain values from $[a, b]$ where $a > b$, and the task is thus to represent vectors from $[a, b]^n$ where n is the number of real-valued variables.

One way to approach this problem is to use a fixed number k of bits to represent each real-valued variable in the vector [2]. Binary strings that represent n real-valued variables would therefore consist of $n \times k$ bits. The k bits for each real-valued variable X_i are interpreted as binary representation of an integer k_i from 0 to $2^k - 1$. The value of X_i for a particular combination of the k bits representing X_i is given by

$$X_i = a + (b - a)\frac{k_i}{2^k - 1}.$$

Before evaluating each binary string, the values of the real-valued variables are obtained, and the resulting real-valued vector is evaluated using the fitness function. One may also consider mapping real-valued vectors to binary strings, where the value of each variable X_i would determine the closest k_i, and, consequently, the most accurate k-bit representation of X_i.

Clearly, the more bits are used to represent each real value, the more accurate the representation becomes. An appropriate value of k depends on the desired accuracy of the solution and the ruggedness of the fitness landscape, but in many cases, one can use between 5 to 30 bits per real variable. The final solutions can be further tuned with a local search method, using for example a variant of the steepest descent/ascent method.

Intuitively, after several iterations of a GA, most variables will have their values concentrated in one or several smaller areas of the interval $[a, b]$. That is why using a mapping function that assumes that the binary strings can represent only equidistant values becomes inefficient, as most of these values will lead to solutions of low quality and will therefore not be used. That is why one may choose to use an adaptive discretization scheme, in which more points are allocated to subintervals that appear in the population most frequently [26, 27, 28, 29].

Binary strings can also be used to encode candidate solutions from other representations, such as permutations, tree graphs, or combinations of several different representations. Although there are certain benefits of mapping a particular representation to binary strings, such as the potential for finding and exploiting various problem regularities, except for the domain of real-valued vectors, practitioners often prefer to use the original representation of the problem to represent candidate

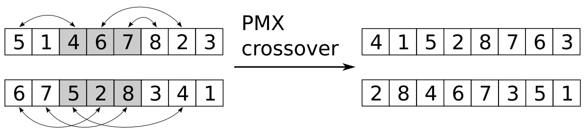


Figure 3: Partially matched crossover starts by selecting two positions at random. Then, the elements between the two positions are matched against the other parent solution and the corresponding pairs of elements define pairs of elements to exchange in each parent solution. In this example, the elements between the two selected positions are $(4, 6, 7)$ and $(5, 2, 8)$; therefore, in both solutions, 4 will exchange with 5, 6 will exchange with 2, and 7 will exchange with 8.

solutions and use modified variation operators capable of directly manipulating solutions in the representation used. Some examples for the domain of permutations and real-valued vectors are presented next.

3.4.2 Variation Operators for Permutations and Real-Valued Vectors

Many important optimization problems deal with candidate solutions represented by permutations; examples of such problems include the quadratic assignment problem and the traveling salesman problem. Although the variation operators presented thus far can be applied to permutations, doing this may often result in candidate solutions that are not valid permutations. One way to deal with this problem is to define a *repair operator*, which corrects candidate solutions after variation so that they become valid. Another approach is to modify the variation operators themselves so that only valid permutations are obtained. Two crossover operators and several mutation operators will be presented next, all of which directly manipulate permutations without making them invalid.

Partially matched crossover (PMX). *Partially matched crossover (PMX)* [30] combines two parent permutations by first selecting two permutation positions at random. Then, for each permutation element between the two positions, the element is exchanged with the corresponding element in the other parent solution. All exchanges are done in both parent solutions. An illustrative example of the partially matched crossover is shown in Figure 3.

Uniform order-based crossover. *Uniform order-based crossover* [31] starts by generating a mask to determine which elements will be copied from the original parent and which will be taken from the other parent permutation. The mask may be defined for example by generating a 1 in each position with probability 50% and generating a 0 otherwise. The two parents retain the elements in the positions where the mask contains a 1. The remaining elements are filled in from the other parent in the order they appear in the other parent. An illustrative example is shown in Figure 4.

Swaps, inversion, and scramble mutation. *Swap mutation* starts by selecting two permutation elements at random and it then exchanges these two elements. *Inversion mutation* starts by selecting two locations in the permutation and it then inverts the permutation elements between these two locations. Finally, *scramble mutation* selects two locations at random and it then randomly reorders all elements between the two selected positions. All three mutation operators are illustrated in Figure 5.

Numerous variation operators have been designed for real-valued vectors. Here we mention a few simple ones.

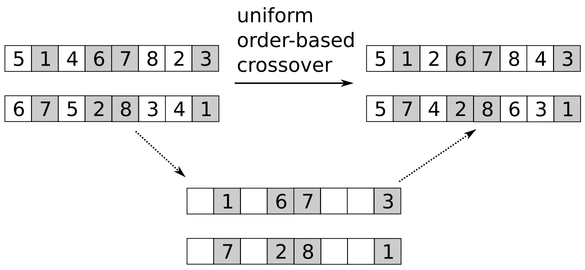


Figure 4: Uniform order-based crossover starts by selecting random positions for which the permutation elements do not change. The remaining positions are filled with the missing elements copied in the order they appear in the other parent.

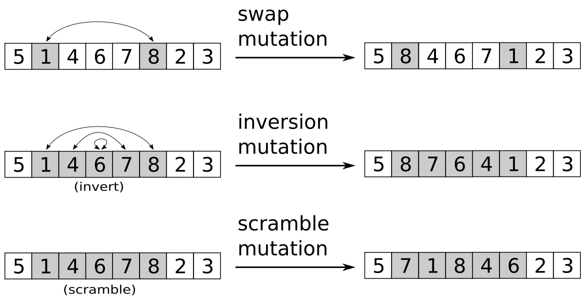


Figure 5: Mutation operators for permutations. All three mutation operators shown here start by randomly selecting two locations in the permutation (in this example, the second and sixth positions are selected). Swap mutation exchanges the elements in the selected locations. Inversion mutation inverts the order of the elements between the two locations. Scramble mutation randomly reorders all the elements between the two locations.

Arithmetic crossover. Arithmetic crossover starts by selecting a location i in the two parents at random and generating a random parameter $\alpha \in [0, 1]$ according to the uniform distribution. Denoting the variable at location i in the first parent by X_i and the corresponding variable in the second parent by Y_i, the values of X_i and Y_i are mapped to X_i' and Y_i', respectively, as follows:

$$X_i' = (1 - \alpha)X_i + \alpha Y_i,$$

$$Y_i' = (1 - \alpha)Y_i + \alpha X_i.$$

Arithmetic crossover can also be applied to several or even all variables.

Simulated binary crossover (SBX). In simulated binary crossover [32], a location for the crossover site is first chosen, similarly as in arithmetic crossover. Let us denote the variables of the two parents for the chosen crossover site by X_i and Y_i, and assume (without loss of generality) that $X_i < Y_i$. The new values of X_i and Y_i are computed as

$$X_i' = \frac{X_i + Y_i}{2} - \frac{\beta}{2}\left(Y_i - X_i\right),$$

$$Y_i' = \frac{X_i + Y_i}{2} + \frac{\beta}{2}\left(Y_i - X_i\right),$$

where $\beta > 0$ is a randomly generated parameter (spread factor). One way to generate β is to use the following probability density function:

$$c(\beta) = \begin{cases} 0.5(n+1)\beta^n & \text{for } \beta \leq 1 \\ 0.5(n+1)\beta^{-n-2} & \text{for } \beta > 1 \end{cases}$$

Similarly as in arithmetic crossover, this procedure can be applied to several or even all variables in the two parents.

Gaussian mutation. In Gaussian mutation, each variable is modified by adding a random number according to a Gaussian distribution with zero mean. The variance of the Gaussian distribution defines the strength of mutation, and is typically significantly smaller than the range of values of a variable.

3.5 Adaptive Operators and Linkage Learning

The combination of selection and variation is the main driving force of the search for the optimum in a GA. Selection selects the most promising candidate solutions whereas variation combines parts of the selected candidate solutions and explores their immediate neighborhood. To ensure that the combination of selection and variation enables robust, efficient and scalable search for the global optimum, it is necessary that variation exploits regularities in the problem landscape and that it combines and modifies best candidate solutions in a meaningful way. Nonetheless, all variation operators discussed in the previous section process candidate solutions in the same way regardless of the specifics of the problem being solved. Consequently, for some classes of difficult problems, practitioners may often have to either modify these variation operators to better suit the problem or modify the representation of candidate solutions to make the existing variation operators more effective. One way to alleviate this problem is to use *adaptive variation operators*, which are capable to adapt to the problem being solved. In this section, we discuss the use of adaptive operators with the main focus on recombination operators capable of *linkage learning* [33, 21, 34, 35].

The previous section pointed out that in one-point crossover implicit linkage is introduced on variables encoded in consequent locations of solution strings. Consequently, combinations of values in consequent positions of solution strings are often preserved by variation, whereas combinations of values in positions located far from each other may be often modified. This has a positive effect on GA performance if the decision variables located close to each other are strongly correlated in the problem being solved. Nonetheless, for many problems variable correlations may appear both between variables that are located close to each other and those that are located far. Furthermore, a practitioner may not be aware of correlations that are important for solving the problem in a scalable manner. That is why practitioners are often forced to define new, problem-specific variation operators or to modify the representation of candidate solutions so that the correlations between problem variables correspond more closely to the implicit linkage introduced by the variation operators used, such as the one-point crossover.

To alleviate this problem, adaptive variation operators can be designed capable of discovering most important interactions between problem variables and modifying the recombination operator accordingly. The process of discovering important interactions between problem variables in order to improve effectiveness of variation is often referred to as *linkage learning* [36, 37, 33, 21, 34, 35]. The term linkage is also common in biological systems, where linkage refers to the level of association in inheritance of two or more non-allelic genes that is higher than that expected if these genes were independent [38].

There are two main approaches to linkage learning [35]. In the *unimetric approach*, linkage learning is based solely on the fitness values obtained during the GA run. In the *multimetric approach*, linkage learning is based on an additional metric besides the fitness function itself. While unimetric approaches appear more biologically plausible, multimetric approaches have been somewhat more successful in practice [35].

GAs capable of linkage learning can be split into three main categories: (1) perturbation techniques, (2) linkage-adaptation techniques and (3) estimation of distribution algorithms. Linkage-adaptation techniques are the only category based on the unimetric approach.

Perturbation techniques. In perturbation-based linkage learning GAs, linkages are identified via perturbing candidate solutions and analyzing the effects of these perturbation on the fitness values. Main representatives of perturbation-based linkage learning GAs are the messy GA [36, 39], the fast messy GA [40, 37], the gene expression messy GA [41], the linkage identification by non-monotonicity detection GA [42, 43], and the dependency-structure-matrix-based GA [44, 45].

Linkage-adaptation techniques. In linkage-adaptation techniques, linkages are evolved along with candidate solutions. Selection thus remains the main driving force of linkage learning. Linkage adaptation is used for example in the linkage-learning GA [46].

Estimation of distribution algorithms. In estimation of distribution algorithms (EDAs) [47, 48, 49, 50]—also called probabilistic model-building genetic algorithms and iterated density estimation algorithms—standard variation operators of genetic algorithms are replaced by building and sampling probabilistic models of selected candidate solutions. Since many classes of probabilistic models enable the discovery and use of dependencies between problem variables, the use of probability distributions provides a powerful tool for multimetric linkage learning. Some of the linkage learning GAs based on the EDA framework are the Bayesian optimization algorithm (BOA) [51, 52], the estimation of Bayesian networks algorithm (EBNA) [53], the learning factorized distribution algorithm (LFDA) [54] and the extended compact genetic algorithm (ecGA) [55]. For more information on EDAs, please see refs. [47, 48, 49, 50].

4 Dealing with Multiple Objectives, Constraints and Noise

Many challenging real-world problems contain multiple objectives or constraints, or the evaluation of solution quality is affected by noise. GAs provide a fairly powerful solution even in these scenarios [56, 57]. In fact, robustness and the ability to deal with multimodal, noisy, constrained and multiobjective optimization problems are among the key advantages of GAs over many other optimization techniques. In this section, we review some of the most important concepts for dealing with problems with multiple objectives, noise, and constraints. The main focus will be on providing the interested reader with pointers.

4.1 Multiobjective Genetic Algorithms

In multiobjective optimization, the task is to maximize or minimize several objectives. For example, in engine design, one may want to maximize performance and minimize fuel consumption. The main difficulty of dealing with multiobjective problems is that the objectives are typically conflicting because increasing solution quality with respect to one objective leads to decreased quality with respect to another objective. There are two basic approaches to solving multiobjective optimization problems using GAs and other evolutionary algorithms [56, 57]:

Transform multiple objectives into a single objective. One way to deal with a multiobjective problem is to transform all objectives into a single objective using a weighted sum, utility theory, or another approach. The main difficulty lies in the appropriate selection of weights or utility functions because there is typically only little problem-specific knowledge to guide these design choices.

Find the entire Pareto-optimal set (tradeoff between objectives). Another approach is to find the entire Pareto-optimal set (front) of solutions or a representative subset of the entire Pareto-optimal set [58, 59]. A set of candidate solutions is called Pareto optimal if the set contains only solutions that are not dominated (outperformed) by any other admissible solution in all objectives. See Figure 6 for an illustration of the Pareto-optimal set. The set of Pareto-optimal solutions defines the trade-off between the objectives and can serve as the main input for finding the solution of desired properties. The main difficulty in finding the entire Pareto-optimal set is that the size of the set typically grows very fast with the number of objectives and so does the difficulty of finding enough representatives of this set. Two most popular variants of a GA for finding the Pareto-optimal set of solutions are (1) the non-dominated sorting GA-II (NSGA-II) [60] and (2) the improved strength Pareto evolutionary algorithm (SPEA2) [61]. In both algorithms, the modifications take place in selection and replacement operators; other operators remain the same as in standard, single-objective GAs.

For an excellent introduction to solving multiobjective optimization problems using GAs, see for example refs. [56, 57].

4.2 Noisy Evaluation

In many real-world problems, evaluation of candidate solutions is affected by noise. The sources of noise include physical measurements, approximate models, stochastic simulation models, incom-

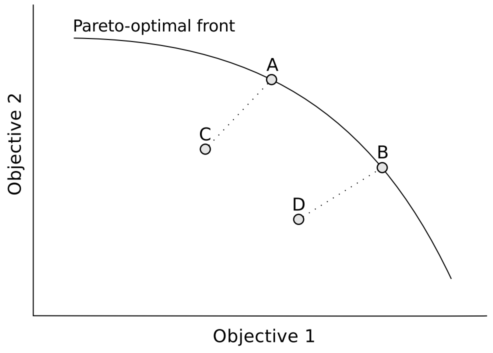


Figure 6: In this example the task is to maximize two objectives and each axis in the graph corresponds to one of these objectives. The two solutions A and B are members of the Pareto-optimal front (set), and they thus cannot be dominated by any other admissible solution in both objectives. Two solutions C and D are not members of the Pareto-optimal set, because they are dominated by other solutions. For example, C is dominated by A and D is dominated by B.

plete information, and human-computer interaction [62]. Genetic and other evolutionary algorithms are known to be capable of dealing with problems including noise particularly well [63, 62]. This is due to several reasons. Most importantly, one of the consequences of using a population of candidate solutions as opposed to a single solution is that the use of a population alleviates the effects of noise by averaging them out. Additionally, GAs do not require derivatives or gradients, which are difficult to approximate in the presence of noise.

One way to alleviate the effects of noise in fitness evaluation is to increase the population size [63, 10, 21]. In addition to the increased population-sizing requirements, in presence of noise, the overall number of iterations until convergence can be expected to increase [63, 21].

Besides increasing the population size and the number of iterations, one may deal with noise by using *fitness averaging*. The main idea of fitness averaging is to evaluate each candidate solution several times, and use the average fitness of these evaluations as the fitness of this solution. Given the available time for the GA and the specifics of the problem, one can compute the optimal number of evaluations for each candidate solution [64].

4.3 Constraints

In many real-world optimization problem, the set of admissible solutions is defined using a set of constraints, which may be defined for example by providing lower bounds for linear combinations of variables. The task is to maximize or minimize a given objective function subject to the given constraints.

There are three main types of approaches to using GAs for solving optimization problems with linear or nonlinear constraints:

Preserve feasibility of candidate solutions. Specialized variation operators may be defined that are always guaranteed to preserve feasibility of candidate solutions. If the initial population is generated so that it contains only feasible solutions and if the specialized variation operators are used, all solutions obtained during the GA run will be guaranteed to be feasible with respect to the given constraints. Assuming that the search space is convex and that the feasible solutions are represented by real-valued vectors, one may use for example the genetic algorithm for numerical optimization of constrained problems (GENOCOP) [65].

Another way to ensure feasibility of candidate solutions is to reject infeasible solutions [66] or to control the proportion of infeasible solutions in some way [67]. Finally, one may introduce *repair* operators, which modify each infeasible solution to make it feasible [68].

Penalize infeasible solutions. If the population is allowed to contain solutions that violate some of the constraints, infeasible solutions may be penalized so that the search is biased toward feasible regions of the search space. This can be done, for example, by introducing a static or dynamic penalty function that reduces fitness of infeasible solutions [69, 70, 71] or by modifying the selection operator to consider constraint violation in addition to the fitness value [72].

Hybrid approaches. The above two classes of approaches to solving constrained optimization problems can be combined in various ways, creating hybrid approaches to dealing with constrained problems.

For an overview of the work on solving optimization problems with constraints using genetic and other evolutionary algorithms, the interested reader should consult refs. [73, 74, 65].

5 Efficiency Enhancement

Practical applications of GAs often necessitate the use of additional efficiency enhancement techniques, such as parallelization and hybridization. There are four main categories of efficiency enhancement techniques: (1) Parallelization, (2) hybridization, (3) evaluation relaxation and (4) time continuation. This section reviews these main types of efficiency enhancement techniques.

5.1 Parallelization

One way to speed up GAs is to distribute the computation to multiple processors [75, 76, 77, 78, 79]. Due to their nature, GAs can be parallelized in a number of ways. There are three main types of parallel GAs [78]: (1) single-population master-slave GAs, (2) single-population fine-grained GAs, and (3) multiple-population coarse-grained GAs (see Figure 7 for an illustration of these types).

Single-population master-slave GAs. In the master-slave architecture, the computation is controlled by a master processor, which stores the population and executes GA operators. Slave processors are typically used only to compute the fitness values of candidate solutions. Other GA operators may also be distributed, but due to the relatively high cost of communication between different processors, it is important to ensure that the savings in computational time are more significant than the time required for communication between the master and slave processors.

Single-population fine-grained GAs. Fine-grained GAs consist of one spatially structured population which is distributed among a large number of processing units. Fine-grained GAs are best suited for massively parallel computers. Ideally, each processing unit considers only one candidate solution. Selection and variation are restricted based on the spatial distribution and communication structure of the parallel computer. For example, the individuals may be distributed on a 2D grid and communication may be restricted to consider only 4 neighbors of each candidate solution.

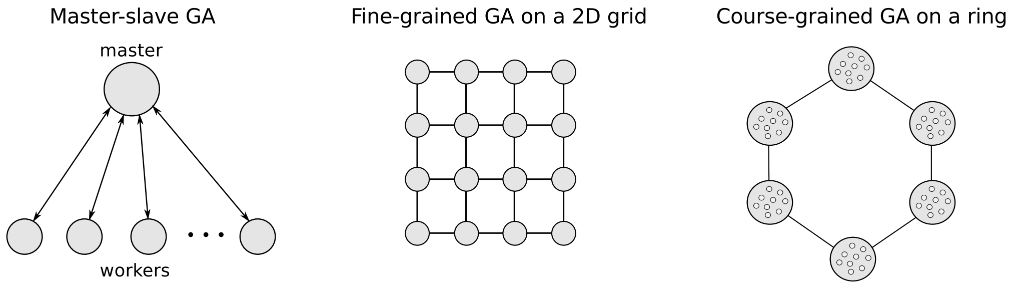


Figure 7: Illustration of different types of parallel GAs.

Multiple-population coarse-grained GAs. The multiple-population coarse-grained GAs split the population into several subpopulations which develop in isolation. Infrequent migration of individuals from one subpopulation to another one is allowed. There are numerous parameters that define the migration patterns, such as when and how many individuals are exchanged between subpopulations, and which subpopulations are allowed to exchange individuals.

Ideally, if the GA is executed in parallel using p processors, the speedup compared to the standard GA implementation on a single processor should be approximately p or even more; otherwise, the computational resources are not used efficiently. For introduction and theoretical guidelines for designing efficient parallel GAs, see for example refs. [76, 79, 80].

5.2 Hybridization

The central idea in hybridization is to combine two or more optimization methods into a single hybrid technique. Typically, global-local hybrids are used which combine two optimization procedures; one of these methods is capable of effective global exploration of the search space whereas the other is capable of quickly reaching a local optimum. In hybrid GAs, a GA would be typically used as a global optimizer and some form of hill climbing would be used as a local searcher. The local searcher may be run on every solution in the population or only on a portion of the population (for example, on the best solution obtained). Hybrid GAs are sometimes referred to as *memetic GAs*. A survey to hybrid GAs can be found for example in ref. [81].

One of the key issues in the design of efficient hybrid procedures is the appropriate coordination of the global and local search. The two main possibilities are captured in the literature by the terms *Baldwinian* and *Lamarckian* hybrid [82]:

Baldwinian hybrid. In a Baldwinian hybrid, the local search procedure uses the global procedure's value as a starting point, searches until reaching an optimum, and then passes back the function value found *without* backsubstituting the solution value found.

Lamarckian hybrid. In a Lamarckian hybrid, the local search procedure again uses the global procedure's value as a starting point, searches until reaching an optimum, and then passes back the function value found *with* backsubstitution of the solution value found.

At the first sight, Lamarckian hybrids may seem more intuitive, but in some cases backsubstitution of solution values found by local search has been found to reduce solution variance to a point such that subsequent exploration is diminished. Various rules of thumb have suggested Baldwinian moves with a small percentage of Lamarckian moves as desirable in practice [68].

Finding an adequate way to split resources between the global and local search requires a systems-level understanding of the roles of the two procedures. Goldberg and Voessner [83] proposed such a theory and Sinha et al. [84, 85, 86] explored the theory in the context of simple GAs. Simply stated, the theory suggests that the key role of the global searcher is to find one or a number of good regions and the key role of the local searcher is to find local optima within these regions.

5.3 Evaluation Relaxation

For many optimization problems, evaluation of the objective function is computationally expensive. The number of evaluations until a solution is found can be substantially decreased using hybridization and prior knowledge, and a number of studies exist that support this fact [34, 87, 88, 54, 21, 89]. However, to deal with computationally expensive objective functions, one may also have to use past evaluations to estimate fitness values for some of the new solutions. There are two basic approaches to using past evaluations to estimate fitness values of new solutions:

Exogenous models. Past evaluations can be used to build a fast, approximate model of the objective function, which can subsequently be used to evaluate some of the new candidate solutions instead of the original, computationally expensive objective function [90, 91, 92, 89]. For example, a simple linear model can be adjusted to fit the fitness values in the population of parents and then sampled to estimate fitness values of some of the new solutions [90]. More advanced models can be developed based on probabilistic models discovered by estimation of distribution algorithms [91, 89] or neural networks [92].

Endogenous models. The fitness of new solutions can also be estimated directly from parental fitness values [93] because new solutions have often fitness values similar to the parent solutions used to create these new solutions. For example, the fitness of a new solution can be estimated as an average fitness of the parents that were used to create this new solution [93].

Using an approximate model to evaluate some candidate solutions has typically a similar effect as if the fitness function was affected by additive external noise [90]. Consequently, the overall number of candidate solutions that must be considered until an accurate solution is obtained can be expected to increase. However, since many of the solutions are evaluated using a fast approximation of the computationally expensive fitness function, the overall computational time can be reduced substantially. In practice, even speedups of over 50 can be obtained if advanced exogenous models are used in combination with model building using estimation of distribution algorithms [91].

5.4 Time Continuation

Different variation operators have different function and effects. Recombination combines bits and pieces between pairs of promising solutions whereas mutation modifies candidate solutions to explore their immediate neighborhood. The optimal division of labor between the two operators depends on the specifics of the problem and the time available to complete the task. Furthermore, depending on the time available and problem specifics, in some situations it may be preferable to search for solutions using a large population whereas in other cases it may be preferable to search for solutions using a small population with multiple convergence epochs. In time continuation [94, 95, 21], the focus is on the optimal division of labor between different search operators and other parameters of a GA with the focus on either (1) maximizing solution quality within a limited computational budget or (2) minimizing computational time for obtaining a solution of given quality.

Theoretical results for time continuation indicate that for additively separable problems with subproblems of equal salience, multiple convergence epochs with a rather small population are expected to be more efficient. However, in presence of noise and overlap between subproblems, a single epoch with a large population is expected to be more efficient. For an overview of work on time continuation and theoretical results, please see refs. [95, 96].

6 Guidelines for Applying Genetic Algorithms to a New Problem

One of the main advantages of GAs over other optimization methods is that it is relatively straightforward to apply GAs to new optimization problems. This section provides guidelines that can serve as a starting point for novices who want to apply a GA to a new problem.

Parameters. There are numerous parameters that must be set in order to apply a GA to a new problem. The simple GA implementation presented in section 2 contains three main parameters (besides the problem-dependent ones). The best values of these three parameters depend on the problem being solved, but there are some general rules of thumb that provide a fairly good starting point:

- *Population size, N.* The population size is probably the most important parameter to tune and using populations that are too small is a frequent mistake that GA researchers and practitioners make. Generally, the population size should increase with problem size (the number of bits or variables), and the larger the population size, the better the obtained solution should be. Unless the evaluation of the objective function is prohibitively slow, the population size can range from tens to thousands or even more. As a starting point for initial runs, the population size may be set to the problem size, $N = n$. If using $N = n$ is infeasible, one may start with smaller values of N, such as $N = 50$. Changing the population size and rerunning the GA with larger and smaller populations (e.g. $N/2$ and $2N$) provides an indication of whether the current population size is adequate. If increasing the population size leads to better solutions, the population size should be increased. If increasing the population size leads to solutions of about the same quality, the current population size is likely to suffice. Of course, if the evaluation of the objective function is computationally intensive, large populations may become intractable and, to run a sufficient number of generations, one is often forced to use a relatively small population size.
- *Crossover probability, p_c.* The probability of crossover in GAs is typically quite large. For example, DeJong [97] suggests that $p_c = 0.6$ regardless of the number of bits n. Many researchers use an even larger probability of crossover, for example, $p_c = 0.9$ or $p_c = 1$.
- *Mutation probability, p_m.* The probability of mutation is typically set so that the expected number of modified bits is fixed regardless of n and that only a few bits are expected to change on average. For example, one may use $p_m = 1/n$. Some researchers use larger values of p_m, but in many cases the reasons for increasing the probability of mutation are that the remaining parameter settings are inadequate.

Providing the GA with adequate values of parameters is a long-standing topic in GA research, and for more information, there are numerous publications that the reader may consult; refs. [98, 21] provide a good starting point for these efforts.

Termination. Typically, each GA run is terminated when the number of generations reaches a predefined threshold, when a solution of sufficient quality has been achieved, or when the GA run starts to stagnate without improving the quality of the population for a given number of iterations. One of the frequent mistakes of GA practitioners is to underestimate the population size and then overestimate the number of generations. Doing this can lead to an unnecessary increase in computational requirements of the GA by executing a large number of GA iterations with no or only little improvement. The number of generations is typically only several times larger than the number of bits in the problem. One way to avoid setting a strict upper bound on the number of generations is to terminate the run when the average fitness of the current population does not improve much for a given number of iterations (e.g. 5 or 10 generations). Of course, the best way to terminate a GA run depends on the problem being solved and several trial runs should provide enough input to make an informed decision.

Representation. Representation of candidate solutions is a critical component of a GA. One may either use the original representation of candidate solutions or one may map the original representation into binary strings, real-valued vectors or another representation. A good starting point is to use the original representation for the particular problem with no significant modifications or transformations. In the initial stage, modifications should be done only if the chosen implementation necessitates them. If the results obtained are not satisfactory, alternative representations should be examined.

Operators. Numerous operators were described in this article and more can be found in the references provided. What operators should one choose? Similarly as for the representations, the best way to start is to try simple operators provided in the literature for the representation used. If one starts with a computer implementation of a GA downloaded from the internet, the implementation most likely includes numerous operators to consider; sticking with the default choices should be a great way to start. If the results are not satisfactory, different operators may be examined and specialized operators may be designed.

Fitness function. The fitness function resides right at the heart of a GA—the fitness function defines what solutions are good and what solutions are bad. Since the fitness function depends completely on the problem being solved, it is difficult to give a set of general guidelines for designing adequate fitness functions. The best way to start in the design of a fitness function is to look at the examples from the specific area.

7 Starting Points for Obtaining Additional Information

Introductory Books and Tutorials

Numerous books and other publications exist that provide introduction to genetic algorithms and additional starting points. The following list of references provides some of them: [1, 99, 2, 31, 100, 101, 102, 103, 24, 104, 105].

Software

The following list provides some of the popular GA implementations available online free of charge. These implementations should provide a good starting point for the interested reader.

- *A C++ Library of Genetic Algorithm Components*
  `http://lancet.mit.edu/ga/`
- *Algorithm::Evolutionary, Perl Library for Evolutionary Computation*
  `http://opeal.sourceforge.net/`
- *Genetic ALgorithm Optimized for Portability and Parallelism System (GALLOPS)*
  `http://garage.cse.msu.edu/software/galopps/index.html`
- *Open BEAGLE, a Versatile Evolutionary Computation Framework*
  `http://beagle.gel.ulaval.ca/`
- *Simple GA Implementation in C (based on ref. [2])*
  `http://www.illigal.uiuc.edu/pub/src/simpleGA/C/`
- *Other GA Software Packages*
  `http://www-2.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/ga/systems/0.html`

Journals

The following journals are key venues for papers on GAs and evolutionary computation, although papers on GAs can be found in many other journals focusing on optimization, artificial intelligence, machine learning, and applications.

- *Evolutionary Computation* (MIT Press)
  `http://www.mitpressjournals.org/loi/evco`
- *Evolutionary Intelligence* (Springer)
  `http://www.springer.com/engineering/journal/12065`
- *Genetic Programming and Evolvable Machines* (Springer)
  `http://www.springer.com/computer/ai/journal/10710`
- *IEEE Transactions on Evolutionary Computation* (IEEE Press)
  `http://ieeexplore.ieee.org/servlet/opac?punumber=4235`
- *Natural Computing* (Springer)
  `http://www.springer.com/computer/theoretical+computer+science/journal/11047`

Conferences

The following conferences provide the most important venues for publishing papers on GAs and evolutionary computation, although similarly as for journals, papers on GAs are often published in other venues.

- *ACM SIGEVO Genetic and Evolutionary Computation Conference (GECCO)*
- *European Workshops on Applications of Evolutionary Computation (EvoWorkshops)*
- *IEEE Congress on Evolutionary Computation (CEC)*
- *Main European Events on Evolutionary Computation (EvoStar)*

- *Parallel Problem Solving in Nature (PPSN)*
- *Simulated Evolution and Learning (SEAL)*

Acknowledgments

This project was sponsored by the National Science Foundation (NSF) under CAREER grant ECS-0547013, and by the Univ. of Missouri in St. Louis through the High Performance Computing Collaboratory sponsored by ITS and the Research Award and Research Board programs. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF, AFOSR, or the U.S. Government.

References

[1] J. H. Holland, *Adaptation in natural and artificial systems.* Ann Arbor, MI: University of Michigan Press, 1975.

[2] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison-Wesley, 1989.

[3] J. Niesse and H. Mayne, "Global geometry optimization of atomic clusters using a modified genetic algorithm in space-fixed coordinates," *Journal of Chemical Physics*, vol. 105, no. 11, pp. 4700–4706, 1996.

[4] K. Sastry, "Efficient atomic cluster optimization using a hybrid extended compact genetic algorithm with seeded population," IlliGAL Report No. 2001018, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2001.

[5] S. J. Louis, "Learning from experience: Case injected genetic algorithm design of combinational logic circuits," in *Proceedings of the Fifth International Conference on Adaptive Computing in Design and Manufacturing* (I. C. Parmee, ed.), pp. 295–306, Springer, 2002.

[6] N. Krasnogor and J. Smith, "A tutorial for competent memetic algorithms: model, taxonomy, and design issues," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 5, pp. 474–488, 2005.

[7] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," *Foundations of Genetic Algorithms*, vol. 1, pp. 69–93, 1991.

[8] E. Cantú-Paz, "Comparing selection methods of evolutionary algorithms using the distribution of fitness," Tech. Rep. UCRL-JC-138582, University of California Lawrence Livermore National Laboratory, 2000.

[9] A. Brindle, *Genetic algorithms for function optimization.* Doctoral dissertation and technical report tr81-2, Department of Computer Science, University of Alberta, Edmonton, 1981.

[10] G. Harik, E. Cantú-Paz, D. E. Goldberg, and B. L. Miller, "The gambler's ruin problem, genetic algorithms, and the sizing of populations," *Evolutionary Computation*, vol. 7, no. 3, pp. 231–253, 1999.

[11] H. Mühlenbein and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization," *Evolutionary Computation*, vol. 1, no. 1, pp. 25–49, 1993.

[12] D. Thierens and D. Goldberg, "Convergence models of genetic algorithm selection schemes," *Parallel Problem Solving from Nature*, pp. 116–121, 1994.

[13] D. E. Goldberg and L. Wang, "Adaptive niching via coevolutionary sharing," in *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science*, ch. 2, pp. 21–38, West Sussex, England: John Wiley & Sons, 1997.

[14] C. K. Oei, D. E. Goldberg, and S.-J. Chang, "Tournament selection, niching, and the preservation of diversity," IlliGAL Report No. 91011, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 1991.

[15] J. Horn, "Finite Markov chain analysis of genetic algorithms with niching," *Proc. of the Int. Conf. on Genetic Algorithms (ICGA-93)*, pp. 110–117, 1993.

[16] S. W. Mahfoud, *Niching methods for genetic algorithms.* Unpublished doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL, 1995.

[17] D. E. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," *Proc. of the Int. Conf. on Genetic Algorithms (ICGA-87)*, pp. 41–49, 1987.

[18] B. Horton, "The restricted breed GA for multi-modal genetic algorithms: Comparison with sequential niching, crowding and sharing." Submitted to Evolutionary Computation, 1995.

[19] K. A. De Jong, *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, Ann Arbor, 1975. (University Microfilms No. 76-9381).

[20] G. R. Harik, "Finding multimodal solutions using restricted tournament selection," *Proc. of the Int. Conf. on Genetic Algorithms (ICGA-95)*, pp. 24–31, 1995.

[21] D. E. Goldberg, *The design of innovation: Lessons from and for competent genetic algorithms*, vol. 7 of *Genetic Algorithms and Evolutionary Computation.* Kluwer Academic Publishers, 2002.

[22] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Stuttgart: Frommann-Holzboog, 1973.

[23] I. Rechenberg, *Evolutionsstrategie '94.* Stuttgart: Frommann-Holzboog Verlag, 1994.

[24] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Handbook of Evolutionary Computation.* Oxford: Oxford University Press, 1997.

[25] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies – a comprehensive introduction," *Natural Computing*, vol. 1, no. 1, pp. 3–52, 2002.

[26] E. Cantú-Paz, "Supervised and unsupervised discretization methods for evolutionary algorithms," *Workshop Proc. of the Genetic and Evolutionary Computation Conf. (GECCO-2001)*, pp. 213–216, 2001.

[27] S. Tsutsui, M. Pelikan, and D. E. Goldberg, "Evolutionary algorithm using marginal histogram models in continuous domain," *Knowledge-Based Intelligent Information Engineering Systems & Allied Thechnologies (KES-2001)*, pp. 112–121, 2001.

[28] M. Pelikan, D. E. Goldberg, and S. Tsutsui, "Combining the strengths of the Bayesian optimization algorithm and adaptive evolution strategies," IlliGAL Report No. 2001023, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2001.

[29] Y. ping Chen and C.-H. Chen, "Enabling the extended compact genetic algorithm for real-parameter optimization by using adaptive discretization," *Evolutionary Computation*, vol. 18, no. 2, pp. 199–228, 2010.

[30] D. E. Goldberg and R. Lingle, Jr., "Alleles, loci, and the traveling salesman problem," *Proc. of the Int. Conf. on Genetic Algorithms and Their Applications*, pp. 154–159, 1985.

[31] L. Davis, *Handbook of Genetic Algorithms.* New York: Van Nostrand Reinhold, 1991.

[32] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Systems*, vol. 9, pp. 115–148, 1995.

[33] G. R. Harik and D. E. Goldberg, "Learning linkage," *Foundations of Genetic Algorithms*, vol. 4, pp. 247–262, 1996.

[34] M. Pelikan, *Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms.* Springer, 2005.

[35] Y. ping Chen, T.-L. Yu, K. Sastry, and D. E. Goldberg, "A survey of genetic linkage learning techniques," IlliGAL Report No. 2007014, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2007.

[36] D. E. Goldberg, B. Korb, and K. Deb, "Messy genetic algorithms: Motivation, analysis, and first results," *Complex Systems*, vol. 3, no. 5, pp. 493–530, 1989.

[37] H. Kargupta, *SEARCH, polynomial complexity, and the fast messy genetic algorithm.* PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 1995.

[38] D. L. Hartl and E. W. Jones, *Genetics: principles and analysis.* Jones and Bartlett Publishers, 4th ed., 1998.

[39] K. Deb, *Binary and floating-point function optimization using messy genetic algorithms.* PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 1991.

[40] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik, "Rapid, accurate optimization of difficult problems using fast messy genetic algorithms," *Proc. of the Int. Conf. on Genetic Algorithms*, pp. 56–64, 1993.

[41] S. Bandyopadhyay, H. Kargupta, and G. Wang, "Revisiting the GEMGA: Scalable evolutionary optimization through linkage learning," *Proc. of the Int. Conf. on Evolutionary Computation (ICEC-98)*, pp. 603–608, 1998.

[42] M. Munetomo and D. E. Goldberg, "Linkage identification bvy non-monotonicity detection for overlapping functions," *Evolutionary Computation*, vol. 7, no. 4, pp. 377–398, 1999.

[43] R. B. Heckendorn and A. H. Wright, "Efficient linkage discovery by limited probing," *Evolutionary Computation*, vol. 12, no. 4, pp. 517–545, 2004.

[44] T.-L. Yu, D. E. Goldberg, and Y.-P. Chen, "A genetic algorithm design inspired by organizational theory: A pilot study of a dependency structure matrix driven genetic algorithm," IlliGAL Report No. 2003007, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2003.

[45] T.-L. Yu, D. E. Goldberg, K. Sastry, C. F. Lima, and M. Pelikan, "Dependency structure matrix, genetic algorithms, and effective recombination," *Evolutionary Computation*, 2009. In press.

[46] G. R. Harik, *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. PhD thesis, University of Michigan, Ann Arbor, 1997.

[47] M. Pelikan, D. E. Goldberg, and F. Lobo, "A survey of optimization by building and using probabilistic models," *Computational Optimization and Applications*, vol. 21, no. 1, pp. 5–20, 2002.

[48] P. Larrañaga and J. A. Lozano, eds., *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Boston, MA: Kluwer, 2002.

[49] M. Pelikan, K. Sastry, and E. Cantú-Paz, eds., *Scalable optimization via probabilistic modeling: From algorithms to applications*. Springer-Verlag, 2006.

[50] J. A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, eds., *Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms*. Springer, 2006.

[51] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "BOA: The Bayesian optimization algorithm," *Genetic and Evolutionary Computation Conference (GECCO-99)*, vol. I, pp. 525–532, 1999.

[52] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "Bayesian optimization algorithm, population sizing, and time to convergence," in *Genetic and Evolutionary Computation Conference (GECCO-2000)* (D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, eds.), (San Francisco, CA), pp. 275–282, Morgan Kaufmann, 2000.

[53] P. Larrañaga, R. Etxeberria, J. Lozano, and J. Pena, "Combinatorial optimization by learning and simulation of Bayesian networks," *Proc. of the Uncertainty in Artificial Intelligence (UAI-2000)*, pp. 343–352, 2000.

[54] H. Mühlenbein and T. Mahnig, "FDA – A scalable evolutionary algorithm for the optimization of additively decomposed functions," *Evolutionary Computation*, vol. 7, no. 4, pp. 353–376, 1999.

[55] G. Harik, "Linkage learning via probabilistic modeling in the ECGA," IlliGAL Report No. 99010, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 1999.

[56] K. Deb, *Multi-objective optimization using evolutionary algorithms*. Chichester, UK: John Wiley & Sons, 2001.

[57] C. A. C. Coello, G. B. Lamont, and D. A. Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*. Genetic and Evolutionary Computation, Springer, 2nd ed., 2007.

[58] C. M. Fonseca and P. J. Fleming, "Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization," in *Proc. of the Int. Conf. on Genetic Algorithms (ICGA-93)* (S. Forrest, ed.), (San Mateo, CA), pp. 416–423, Morgan Kaufmann, 1993.

[59] J. Horn and N. Nafpliotis, "Multiobjective optimization using the niched pareto genetic algorithm," IlliGAL Report No. 93005, University of Illinois at Urbana-Champaign, Urbana, IL, July 1993.

[60] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[61] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm," TIK-Report 103, Department of Electrical Engineering, Swiss Federal Institute of Technology (ETH) Zurich, Zurich, Switzerland, 2001.

[62] D. Arnold, *Noisy Optimization with Evolution Strategies.* Genetic Algorithms and Evolutionary Computation, Kluwer, 2002.

[63] B. L. Miller and D. E. Goldberg, "Genetic algorithms, selection schemes, and the varying effects of noise," *Evolutionary Computation*, vol. 4, no. 2, pp. 113–131, 1996.

[64] B. L. Miller and D. E. Goldberg, "Optimal sampling for genetic algorithms," *Intelligent Engineering Systems through Artificial Neural Networks*, vol. 6, pp. 291–297, 1996.

[65] Z. Michalewicz and C. Z. Janikow, "Handling constraints in genetic algorithms," *Proc. of the Int. Conf. on Genetic Algorithms (ICGA-91)*, pp. 151–157, 1991.

[66] T. Baeck, F. Hoffmeister, and H.-P. Schwefel, "A survey of evolution strategies," *Proc. of the Int. Conf. on Genetic Algorithms (ICGA-91)*, pp. 2–9.

[67] M. Schoenauer and S. Xanthakis, "Constrained ga optimization," *Proc. of the Int. Conf. on Genetic Algorithms (ICGA-93)*, pp. 573–580.

[68] D. Orvosh and L. Davis, "Shall we repair? genetic algorithms, combinatorial optimization, and feasibility constraints," *Proceedings of the 5th International Conference on Genetic Algorithms*, p. 650, 1993.

[69] "Using genetic algorithms in engineering design optimization with non-linear constraints," *Proc. of the Int. Conf. on Genetic Algorithms (ICGA-93)*, pp. 424–430, 1993.

[70] A. Homaifar, C. X. Qi, and S. H. Lai, "Constrained optimization via genetic algorithms," *Simulation*, vol. 62, no. 4, pp. 242–254, 1994.

[71] J. A. Joines and C. R. Houck, "On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with gas," *Proc. of the Int. Conf. on Evolutionary Computation (ICEC-94)*, pp. 579–584, 1994.

[72] K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 2-4, pp. 311–338, 2000.

[73] C. A. C. Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art," *Computer Methods in Applied Mechanics and Engineering*, vol. 191, no. 11-12, pp. 1245–1287, 2002.

[74] Z. Michalewicz, “A survey of constraint handling techniques in evolutionary computation methods,” *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pp. 135–155, 1994.

[75] J. P. Cohoon, S. U. Hegde, W. N. Martin, and D. Richards, “Punctuated equilibria: A parallel genetic algorithm,” *Proc. of the Int. Conf. on Genetic Algorithms (ICGA-87)*, pp. 148–154, 1987.

[76] D. E. Goldberg, “Sizing populations for serial and parallel genetic algorithms,” *Proc. of the Int. Conf. on Genetic Algorithms (ICGA-89)*, pp. 70–79, 1989.

[77] H. Mühlenbein, “Evolution in time and space-The parallel genetic algorithm,” *Foundations of Genetic Algorithms*, pp. 316–337, 1991.

[78] E. Cantú-Paz, “A survey of parallel genetic algorithms,” *Calculateurs Paralleles, Reseaux et Systems Repartis*, vol. 10, no. 2, pp. 141–171, 1998.

[79] E. Cantú-Paz, *Efficient and Accurate Parallel Genetic Algorithms.* Boston, MA: Kluwer, 2000.

[80] E. Cantú-Paz, “Efficient parallel genetic algorithms: theory and practice,” *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 4, pp. 221–238, 2000.

[81] A. Sinha and D. E. Goldberg, “A survey of hybrid genetic and evolutionary algorithms,” IlliGAL Report No. 2002XXX, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2002.

[82] G. E. Hinton and S. J. Nowlan, “How learning can guide evolution,” *Complex Systems*, vol. 1, pp. 495–502, 1987.

[83] D. E. Goldberg and S. Voessner, “Optimizing global-local search hybrids,” *Genetic and Evolutionary Computation Conference (GECCO-99)*, pp. 220–228, 1999.

[84] A. Sinha and D. E. Goldberg, “Verification and extension of the theory of global-local hybrids,” *Genetic and Evolutionary Computation Conference (GECCO-2001)*, pp. 591–597, 2001.

[85] A. Sinha, “Designing efficient genetic and evolutionary hybrids,” Master’s thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 2003. Also IlliGAL Report No. 2003020.

[86] A. Sinha, Y.-P. Chen, and D. E. Goldberg, “Designing efficient genetic and evolutionary algorithm hybrids,” in *Recent Advances in Memetic Algorithms* (W. E. Hart, N. Krasnogor, and J. Smith, eds.), pp. 259–288, 2004.

[87] M. Pelikan and A. K. Hartmann, “Searching for ground states of Ising spin glasses with hierarchical BOA and cluster exact approximation,” in *Scalable optimization via probabilistic modeling: From algorithms to applications* (E. Cantú-Paz, M. Pelikan, and K. Sastry, eds.), pp. 333–349, Springer, 2006.

[88] E. Radetic, M. Pelikan, and D. E. Goldberg, “Effects of a deterministic hill climber on hBOA,” *Genetic and Evolutionary Computation Conference (GECCO-2009)*, pp. 437–444, 2009.

[89] K. Sastry, M. Pelikan, and D. E. Goldberg, "Efficiency enhancement of estimation of distribution algorithms," in *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications* (M. Pelikan, K. Sastry, and E. Cantú-Paz, eds.), pp. 161–185, Springer, 2006.

[90] K. Sastry, D. E. Goldberg, and M. Pelikan, "Don't evaluate, inherit," *Genetic and Evolutionary Computation Conference (GECCO-2001)*, pp. 551–558, 2001.

[91] M. Pelikan and K. Sastry, "Fitness inheritance in the Bayesian optimization algorithm," *Genetic and Evolutionary Computation Conference (GECCO-2004)*, vol. 2, pp. 48–59, 2004.

[92] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Computing*, vol. 9, no. 1, pp. 3–12, 2005.

[93] R. E. Smith, B. A. Dike, and S. A. Stegmann, "Fitness inheritance in genetic algorithms," *Proc. of the ACM Symposium on Applied Computing*, pp. 345–350, 1995.

[94] D. E. Goldberg, "Using time efficiently: Genetic-evolutionary algorithms and the continuation problem," *Genetic and Evolutionary Computation Conference (GECCO-99)*, pp. 212–219, 1999.

[95] R. P. Srivastava and D. E. Goldberg, "Verification of the theory of genetic and evolutionary continuation," *Genetic and Evolutionary Computation Conference (GECCO-2001)*, pp. 551–558, 2001.

[96] R. P. Srivastava, "Time continuation in genetic algorithms," Master's thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 2002.

[97] K. A. DeJong and W. M. Spears, "An analysis of the interacting roles of population size and crossover in genetic algorithms," *Parallel Problem Solving from Nature*, pp. 38–47, 1990.

[98] F. G. Lobo, C. F. Lima, and Z. Michalewicz, eds., *Parameter Setting in Evolutionary Algorithms.* Studies in Computational Intelligence Series, Springer, 2007.

[99] L. D. Davis, ed., *Genetic Algorithms and Simulated Annealing.* London: Pitman, 1987.

[100] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, pp. 65–85, 1994.

[101] C. R. Reeves, "Genetic algorithms," in *Modern Heuristic Techniques for Combinatorial Problems* (C. R. Reeves, ed.), New York: McGraw-Hill, 1995.

[102] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs.* Berlin: Springer, 3rd ed., 1996.

[103] M. Mitchell, *An Introduction to Genetic Algorithms.* Cambridge, MA: MIT Press, 1996.

[104] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing.* Springer, 2003.

[105] K. Sastry, D. E. Goldberg, and G. Kendall, "Genetic algorithms: A tutorial," in *Introductory Tutorials in Optimization, Search and Decision Support Methodologies*, ch. 4, pp. 97–125, Springer, 2005.