1
Genetic Algorithms for Optimization

Kalyanmoy Deb
Kanpur Genetic Algorithms Laboratory (KanGAL)
Department of Mechanical Engineering
Indian Institute of Technology Kanpur
Kanpur, PIN 208 016, India
E-mail: deb@iitk.ac.in

1.1 Introduction

A genetic algorithm (GA) is a search and optimization method which works by mimicking the evolutionary principles and chromosomal processing in natural genetics. A GA begins its search with a random set of solutions usually coded in binary string structures. Every solution is assigned a fitness which is directly related to the objective function of the search and optimization problem. Thereafter, the population of solutions is modified to a new population by applying three operators similar to natural genetic operators—reproduction, crossover, and mutation. A GA works iteratively by successively applying these three operators in each generation till a termination criterion is satisfied. Over the past one decade and more, GAs have been successfully applied to a wide variety of problems, because of their simplicity, global perspective, and inherent parallel processing.

Classical search and optimization methods demonstrate a number of difficulties when faced with complex problems. The major difficulty arises when a one algorithm is applied to solve a number of different problems. This is because each classical method is designed to solve only a particular class of problems efficiently. Thus, these methods do not have the breadth to solve different types of problems often faced by designers and practitioners. Moreover, most classical methods do not have the global perspective and often get converged to a locally optimal solution. Another difficulty is their inability to be used in parallel computing environment efficiently. Since most classical algorithms are serial in nature, not much advantage (or speed-up) can be achieved with them.

The GA technique was first conceived by Professor John Holland of University of Michigan, Ann Arbor in 1965. His first book appeared in 1975 (Holland, 1975) and till 1985, GAs have been practiced mainly by Holland and his students (Bagley, 1967; Bethke, 1981; Cavicchio, 1971; De Jong, 1975; Goldberg, 1983). Exponentially more number of researchers and practitioners became interested in GAs soon after the first International conference on GAs held in 1985. Now, there exist a number of books (Gen and Cheng, 1997; Goldberg, 1989; Michalewicz, 1992; Mitchell, 1996) and a few journals dedicated to publishing research papers on the topic (including one from MIT Press and one from IEEE). Every year, there are at least 15-20 conferences and workshops being held on the topic at various parts of the globe. The major reason for GA's popularity in various search and optimization problems is its global perspective, wide spread applicability, and inherent parallelism.

In the remainder of the paper, we shall discuss the working principle of a GA by showing a hand simulation. Thereafter, we argue intuitive reasons for the working of a GA. Later, we discuss several different analytical tools used to understand the complex dynamics of GAs. Finally, the paper briefly mentions a number of extensions which can be used in solving various types of search and optimization problems.

1.2 Classical Search and Optimization Techniques

Traditional search and optimization methods can be classified into two distinct groups: Direct and gradient-based methods (Deb, 1995; Reklaitis at el., 1983). In direct methods, only objective function and constraints are used to guide the search strategy, whereas gradient-based methods use the first and/or second-order derivatives of the objective function and/or constraints to guide the search process. Since derivative information is not used, the direct search methods are usually slow, requiring many function evaluations for convergence. For the same reason, they can be applied to many problems without a major change of the algorithm. On the other hand, gradient-based methods quickly converge to an optimal solution, but are not efficient in non-differentiable or discontinuous problems. In addition, there are some common difficulties with most of the traditional direct and gradient-based techniques:

- Convergence to an optimal solution depends on the chosen initial solution.
- Most algorithms tend to get *stuck* to a suboptimal solution.
- An algorithm efficient in solving one search and optimization problem may not be efficient in solving a different problem.
- Algorithms are not efficient in handling problems having discrete variables.
- Algorithms cannot be efficiently used on a parallel machine.

Because of the nonlinearities and complex interactions among problem variables often exist in complex search and optimization problems, the search space may have many optimal solutions, of which most are locally optimal solutions having inferior objective function values. When solving these problems, if traditional methods get attracted to any of these locally optimal solutions, there is no escape from it.

Many traditional methods are designed to solve a specific type of search and optimization problems. For example, geometric programming (GP) method is designed to solve only posynomial-type objective function and constraints (Duffin et al., 1967). GP is efficient in solving such problems but can not be applied suitably to solve other types of functions. Conjugate direction method has a convergence proof for solving quadratic functions, but they are not expected to work well in problems having multiple optimal solutions. Frank-Wolfe method (Reklaitis at el., 1983) works efficiently on linear-like function and constraints, but the performance largely depends on the chosen initial conditions. Thus, one algorithm may be best suited for one problem and may not be even applicable to a different problem. This requires designers to know a number of optimization algorithms.

In many search and optimization problems, problem variables are often restricted to take discrete values only. To solve such problems, an usual practice is to assume that the problem variables are real-valued. A classical method can then be applied to find a real-valued solution. To make this solution feasible, the nearest allowable discrete solution is chosen. But, there are a number of difficulties with this approach. Firstly, since many infeasible values of problem variables are allowed in the optimization process, the optimization algorithm is likely to take many function evaluations before converging, thereby making the search effort inefficient. Secondly, for each infeasible discrete variable, two values (the nearest lower and upper available sizes) are to be checked. For N discrete variables, a total of 2^N such additional solutions need to be evaluated. Thirdly, two options checked for each variable may not guarantee the optimal combination of all variables. All these difficulties can be eliminated if *only* feasible values of the variables are allowed during the optimization process.

Many search and optimization problems require use of a simulation software, involving finite element technique, computational fluid mechanics approach, solution of nonlinear equations, and others, to compute the objective function and constraints. The use of such softwares is time-consuming and may require several minutes to hours to evaluate one solution. Because of the availability of parallel computing machines, it becomes now convenient to use parallel machines in solving complex search and optimization problems. Since most traditional methods use point-by-point approach, where one solution gets updated to a new solution in one iteration, the advantage of parallel machines cannot be exploited.

The above discussion suggests that traditional methods are not good candidates for an efficient search and optimization algorithm. In the following section, we describe the genetic algorithm which works according to principles of natural genetics and evolution, and which has been demonstrated to solve various search and optimization problems.

1.3 Motivation from Nature

Most biologists believe that the main driving force behind the natural evolution is the Darwin's survival-of-the-fittest principle (Dawkins, 1976; Eldredge, 1989). In most situations, the nature ruthlessly follows two simple principles:

1. If by genetic processing an above-average offspring is created, it is going to survive longer than an average individual and thus have more opportunities to produce children having some of its traits than an average individual.

2. If, on the other hand, a below-average offspring is created, it does not survive longer and thus gets eliminated from the population.

The renowned biologists Richard Dawkins explains many evolutionary facts with the help of Darwin's survival-of-the-fittest principle in his seminal works (Dawkins, 1976; 1986). He argues that the tall trees that exist in the mountains were only a feet tall during early ages of evolution. By genetic processing if one tree had produced an offspring an inch taller than all other trees, that offspring enjoyed more sunlight and rain and attracted more insects for pollination than all other trees. With extra benefits, that lucky offspring had an increased life and more importantly had produced more offspring like it (with tall feature) than others. Soon enough, it occupies most of the mountain with trees having its genes and the competition for survival now continues with other trees, since the available resource (land) is limited. On the other hand, if a tree had produced an offspring with an inch smaller than others, it was less fortunate to enjoy all the facilities other neighboring trees had enjoyed. Thus, that offspring could not survive longer. In a genetic algorithm, this feature of natural evolution is introduced through its operators.

The principle of emphasizing good solutions and deleting bad solutions is a nice feature a population-based approach should have. But one may wonder about the real connection between an optimization procedure and natural evolution! Has the natural evolutionary process tried to maximize a utility function of some sort? Truly speaking, one can imagine a number of such functions which the nature may be thriving to maximize: Life span of a species, quality of life of a species, physical growth, and others. However, any of these functions is non-stationary in nature and largely depends on the evolution of other related species. Thus, in essence, the nature has been really optimizing much more complicated objective functions by means of natural genetics and natural selection than search and optimization problems we are interested in solving. A genetic algorithm is an abstraction of the complex natural genetics and natural selection process. The simple version of a GA described in the following section aims to solve stationary search and optimization problems. Although a GA is a simple abstraction, it is robust and has been found to solve various search and optimization problems of science, engineering, and commerce.

1.4 Genetic Algorithms

In this section, we first describe the working principle of a genetic algorithm. Thereafter, we shall show a simulation of a genetic algorithm for one iteration on a simple optimization problem. Later, we shall give intuitive reasoning of why a GA is a useful search and optimization procedure.

1.4.1 Working Principle

Genetic algorithm (GA) is an iterative optimization procedure. Instead of working with a single solution in each iteration, a GA works with a number of solutions (collectively known as a population) in each iteration. A flowchart of the working principle of a simple GA is shown in Figure 1.1. In the absence of any knowledge of the problem domain, a GA begins its search from a random population of solutions. As shown in the figure, a solution in a GA is represented using a string coding of fixed length. We shall discuss about the details of the coding procedure a little later. But for now notice how a GA processes these strings in a iteration. If a termination criterion is not satisfied, three different operators—reproduction, crossover, and mutation—are applied to update the population of strings. One iteration of these three operators is known as a generation in the parlance of GAs. Since the representation of a solution in a GA is similar to a natural chromosome and GA operators are similar to genetic operators, the above procedure is named as genetic algorithm. We now discuss the details of the coding representation of a solution and GA operators in details in the following subsections.

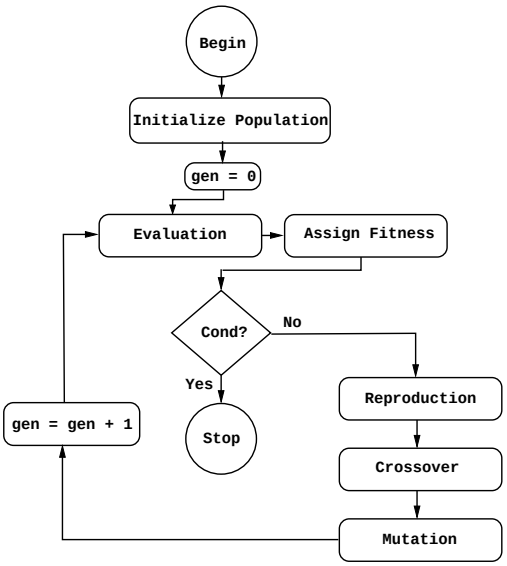

Figure 1.1: A flowchart of working principle of a genetic algorithm

Representation

In a binary-coded GA, every variable is first coded in a fixed-length binary string. For example, the following is a string, representing N problem variables:

$$\underbrace{11010}_{x_1}\ \underbrace{1001001}_{x_2}\ \underbrace{010}_{x_3}\ \cdots\ \underbrace{0010}_{x_N}$$

The i-th problem variable is coded in a binary substring of length ℓ_i, so that the total number of alternatives allowed in that variable is 2^{ℓ_i}. The lower bound solution $x_i^{\min}$ is represented by the solution (00...0) and the upper bound solution $x_i^{\max}$ is represented by the solution (11...1). Any other substring s_i decodes to a solution x_i as follows:

$$x_i = x_i^{\min} + \frac{x_i^{\max} - x_i^{\min}}{2^{\ell_i} - 1}\mathrm{DV}(s_i), \tag{1.1}$$

where $\mathrm{DV}(s_i)$ is the decoded value[1] of the substring s_i. The length of a substring is usually decided by the precision needed in a variable. For example, if three decimal places of accuracy is needed in the i-th variable, the total number of alternatives in the variable must be $(x_i^{\max} - x_i^{\min})/0.001$, which can be set equal to 2^{ℓ_i} and ℓ_i can be computed as follows:

$$\ell_i = \log_2\left(\frac{x_i^{\max} - x_i^{\min}}{\epsilon_i}\right). \tag{1.2}$$

Here, the parameter ϵ_i is the desired precision in the i-th variable. The total string length of a N-variable solution is then $\ell = \sum_{i=1}^{N}\ell_i$. Representing a solution in a string of bits (0 or 1) resembles a natural chromosome which is a collection of genes having particular allele values.

In the initial population, ℓ-bit strings are created at random (at each of ℓ positions, there is a equal probability of creating a 0 or a 1). Once such a string is created, the first ℓ_1 bits can be extracted from the complete string and corresponding value of the variable x_1 can be calculated using Equation 1.1 and using the chosen lower and upper limits of the variable x_1. Thereafter, the next ℓ_2 bits can be extracted from the original string and the variable x_2 can be calculated. This process can be continued until all N variables are obtained from the complete string. Thus, an ℓ-bit string represents a complete solution specifying all N variables uniquely. Once these values are known, the objective function $f(x_1, \ldots, x_N)$ can be computed.

In a GA, each string created either in the initial population or in the subsequent generations must be assigned a *fitness* value which is related to the objective function value. For maximization problems, a string's fitness can be equal to the string's objective function value. However, for minimization problems, the goal is

[1]The decoded value of a binary substring $S \equiv (S_{\ell-1}S_{\ell-2}\ldots S_2S_1S_0)$ is calculated as $\sum_{j=0}^{\ell} -12^j S_j$, where $S_j \in (0,1)$.

to find a solution having the minimum objective function value. Thus, the fitness can be calculated as the reciprocal of the objective function value so that solutions with smaller objective function value get larger fitness. Usually, the following transformation function is used for minimization problems:

$$\text{Fitness} = \frac{1}{1 + f(x_1, \ldots, x_N)}. \tag{1.3}$$

There are a number of advantages of using a string representation to code variables. First, this allows a shielding between the working of GA and the actual problem. What GA processes is ℓ-bit strings, which may represent any number of variables, depending on the problem at hand. Thus, the same GA code can be used for different problems by only changing the definition of coding a string. This allows a GA to have a wide spread applicability. Second, a GA can exploit the similarities in string coding to make its search faster, a matter which is important in the working of a GA and is discussed in Subsection 4.3.

Reproduction

Reproduction (or selection) is usually the first operator applied on a population. Reproduction selects good strings in a population and forms a mating pool. There exists a number of reproduction operators in the GA literature (Goldberg and Deb, 1991), but the essential idea is that above-average strings are picked from the current population and duplicates of them are inserted in the mating pool. The commonly-used reproduction operator is the *proportionate* selection operator, where a string in the current population is selected with a probability proportional to the string's fitness. Thus, the i-th string in the population is selected with a probability proportional to f_i. Since the population size is usually kept fixed in a simple GA, the cumulative probability for all strings in the population must be one. Therefore, the probability for selecting i-th string is $f_i / \sum_{j=1}^{N} f_j$, where N is the population size. One way to achieve this proportionate selection is to use a roulette-wheel with the circumference marked for each string proportionate to the string's fitness. The roulette-wheel is spun N times, each time keeping an instance of the string, selected by the roulette-wheel pointer, in the mating pool. Since the circumference of the wheel is marked according to a string's fitness, this roulette-wheel mechanism is expected to make $f_i/\bar{f}$ copies of the i-th string, where $\bar{f}$ is the average fitness of the population. This version of roulette-wheel selection is somewhat noisy; other more stable versions exist in the literature (Goldberg, 1989). As will be discussed later, the proportionate selection scheme is inherently slow. One fix-up is to use a *ranking* selection scheme. All N strings in a population is first ranked according to ascending order of string's fitness. Each string is then assigned a rank from 1 (worst) to N (best) and an linear fitness function is assigned for all the strings so that the best string gets two copies and the worst string gets no copies after reproduction. Thereafter, the proportionate selection is used with these fitness values. This ranking reproduction scheme eliminates the function-dependency which exists in the proportionate reproduction scheme.

The *tournament* selection scheme is getting increasingly popular because of its simplicity and controlled takeover property (Goldberg and Deb, 1991). In its simplest form (binary tournament selection), two strings are chosen at random for a tournament and the better of the two is selected according to the string's fitness value. If done systematically, the best string in a population gets exactly two copies in the mating pool. It is important to note that this reproduction operator does not require a transformation of the objective function to calculate fitness of a string as suggested in Equation 1.3 for minimization problems. The better of two strings can be judged by choosing the string with the smaller objective function value.

Crossover

Crossover operator is applied next to the strings of the mating pool. Like reproduction operator, there exists a number of crossover operators in the GA literature (Spears and De Jong, 1991; Syswerda, 1989), but in almost all crossover operators, two strings are picked from the mating pool at random and some portion of the strings are exchanged between the strings. In a single-point crossover operator, both strings are cut at an arbitrary place and the right-side portion of both strings are swapped among themselves to create two new strings, as illustrated in the following:

Parent1	0	0	0	0	0	$\Rightarrow$	0	0	1	1	1	Child1
Parent2	1	1	1	1	1		1	1	0	0	0	Child2

It is interesting to note from the construction that good substrings from either parent string can be combined to form a better child string if an appropriate site is chosen. Since the knowledge of an appropriate site

is usually not known, a random site is usually chosen. However, it is important to realize that the choice of a random site does not make this search operation random. With a single-point crossover on two ℓ-bit parent strings, the search can only find at most $2(\ell - 1)$ different strings in the search space, whereas there are a total of 2^ℓ strings in the search space. With a random site, the children strings produced may or may not have a combination of good substrings from parent strings depending on whether the crossing site falls in the appropriate place or not. But we do not worry about this aspect too much, because if good strings are created by crossover, there will be more copies of them in the next mating pool generated by the reproduction operator. But if good strings are not created by crossover, they will not survive beyond next generation, because reproduction will not select bad strings for the next mating pool.

In a *two-point* crossover operator, two random sites are chosen and the contents bracketed by these sites are exchanged between two parents. This idea can be extended to create a multi-point crossover operator and the extreme of this extension is what is known as a *uniform* crossover operator (Syswerda, 1989). In a uniform crossover for binary strings, each bit from either parent is selected with a probability of 0.5.

It is worthwhile to note that the purpose of the crossover operator is two-fold. The main purpose of the crossover operator is to search the parameter space. Other aspect is that the search needs to be performed in a way to preserve the information stored in the parent strings maximally, because these parent strings are instances of good strings selected using the reproduction operator. In the single-point crossover operator, the search is not extensive, but the maximum information is preserved from parent to children. On the other hand, in the uniform crossover, the search is very extensive but minimum information is preserved between parent and children strings. However, in order to preserve some of the previously-found good strings, not all strings in the population are participated in the crossover operation. If a crossover probability of p_c is used then $100p_c\%$ strings in the population are used in the crossover operation and $100(1-p_c)\%$ of the population are simply copied to the new population. Even though best $100(1-p_c)\%$ of the current population can be copied deterministically to the new population, this is usually performed stochastically.

Mutation

Crossover operator is mainly responsible for the search aspect of genetic algorithms, even though mutation operator is also used for this purpose sparingly. Mutation operator changes a `1` to a `0` and vice versa with a small mutation probability, p_m:

`00000` $\Rightarrow$ `00010`

In the above example, the fourth gene has changed its value, thereby creating a new solution. The need for mutation is to maintain diversity in the population. For example, if in a particular position along the string length all strings in the population have a value `0`, and a `1` is needed in that position to obtain the optimum or a near-optimum solution, then the crossover operator described above will be able to create a `1` in that position. The inclusion of mutation introduces some probability of turning that `0` into a `1`. Furthermore, for local improvement of a solution, mutation is useful.

After reproduction, crossover, and mutation are applied to the whole population, one generation of a GA is completed. These three operators are simple and straightforward. Reproduction operator selects good strings and crossover operator recombines good substrings from two good strings together to hopefully form a better substring. Mutation operator alters a string locally to hopefully create a better string. Even though none of these claims are guaranteed and/or tested while creating a new population of strings, it is expected that if bad strings are created they will be eliminated by the reproduction operator in the next generation and if good strings are created, they will be emphasized. Later, we shall discuss some intuitive reasoning as to why a GA with these simple operators may constitute a potential search algorithm.

1.4.2 A Hand Calculation

The working principle described above is simple, with GA operators involving string copying and substring exchange, plus the occasional alteration of bits. Indeed, it is surprising that with such simple operators and mechanisms, a potential search is possible. We will try to give an intuitive answer to such doubts and also remind the reader that a number of studies have attempted to find a rigorous mathematical convergence proof for GAs (Rudolph, 1994; Vose, 1999; Whitley, 1992). Even though the operations are simple, GAs are highly nonlinear, massively multi-faceted, stochastic and complex. There exist studies using Markov chain analysis which involve deriving transition probabilities from one state to another and manipulating them to find the convergence time and solution. Since the number of possible states for a reasonable string length and population size become unmanageable even with the high-speed computers available today, other analytical

Table 1.1: One generation of a GA hand-simulation on the function $\sin(x)$.

Initial population						
String	DV[a]	x	$f(x)$	f_i/f_{avg}	AC[b]	Mating pool
01001	9	0.912	0.791	1.39	1	01001
10100	20	2.027	0.898	1.58	2	10100
00001	1	0.101	0.101	0.18	0	10100
11010	26	2.635	0.485	0.85	1	11010
	Average, f_{avg}		0.569			

Mating Pool	CS[c]	New population			
		String	DV[a]	x	$f(x)$
01001	3	01000	8	0.811	0.725
10100	3	10101	21	2.128	0.849
10100	2	10010	18	1.824	0.968
11010	2	11100	28	2.838	0.299
			Average, f_{avg}		0.710

[a] DV, decoded value of the string.
[b] AC, actual count of strings in the population.
[c] CS, cross site.

techniques (statistical mechanics approaches and dynamical systems models) have also been used to analyze the convergence properties of GAs.

In order to investigate why GAs work, let us apply the GA for only one-cycle to a numerical maximization problem (Deb, 2001):

$$\left.\begin{array}{rl} \text{Maximize} & \sin(x), \\ \text{Variable bound} & 0 \leq x \leq \pi. \end{array}\right\} \tag{1.4}$$

We will use five-bit strings to represent the variable x in the range $[0, \pi]$, so that the string (00000) represents the $x = 0$ solution and the string (11111) represents the $x = \pi$ solution. The other 30 strings are mapped in the range $[0, \pi]$ uniformly. Let us also assume that we use a population of size four, the proportionate selection, the single-point crossover operator with $p_c = 1$, and no mutation (or, $p_m = 0$). To start the GA simulation, we create a random initial population, evaluate each string, and then use three GA operators, as shown in Table 1.1. The first string has a decoded value equal to 9 and this string corresponds to a solution $x = 0.912$, which has a function value equal to $\sin(0.912) = 0.791$. Similarly, the other three stings are also evaluated. Since the proportionate reproduction scheme assigns a number of copies according to a string's fitness, the expected number of copies for each string is calculated in column 5. When the proportionate selection operator is actually implemented, the number of copies allocated to the strings is shown in column 6. Column 7 shows the mating pool. It is noteworthy that the third string in the initial population has a fitness which is very small compared to the average fitness of the population and is eliminated by the selection operator. On the other hand, the second string, being a good string, made two copies in the mating pool. The crossover sites are chosen at random and the four new strings created after crossover is shown in column 3 of the bottom table. Since no mutation is used, none of the bits are altered. Thus, column 3 of the bottom table represents the population at the end of one cycle of a GA. Thereafter, each of these stings is then decoded, mapped and evaluated. This completes one generation of a GA simulation. The average fitness of the new population is found to be 0.710, i.e. an improvement from that in the initial population. It is interesting to note that even though all operators used random numbers, a GA with all three operators produces a directed search, which usually results in an increase in the average quality of solutions from one generation to the next.

1.4.3 Understanding How GAs work

The string copying and substring exchange are all interesting and seem to improve the average performance of a population, but let us now ask the question: 'What has been processed in one cycle of a GA?' If we

investigate carefully, we observe that among the strings of the two populations there are some similarities in the string positions among the strings. By the application of three GA operators, the number of strings with similarities at certain string positions has been increased from the initial population to the new population. These similarities are called *schema* in the GA literature. More specifically, a schema represents a set of strings with certain similarities at certain string positions. To represent a schema for binary codings, a triplet (1, 0 and $*$) is used; a $*$ represents both 1 or 0. It is interesting to note that a string is also a schema representing only one string – the string itself.

Two definitions are associated with a schema. The *order* of a schema H is defined as the number of defined positions in the schema and is represented as $o(H)$. A schema with full order $o(H) = \ell$ represents a string. The *defining length* of a schema H is defined as the distance between the outermost defined positions. For example, the schema $H = (* \ 1 \ 0 \ * \ * \ 0 \ * \ * \ *)$ has an order $o(H) = 3$ (there are three defined positions: 1 at the second gene, 0 at the third gene, and 0 at the sixth gene) and a defining length $\delta(H) = 6 - 2 = 4$.

A schema $H_1 = (1 \ 0 \ * \ * \ *)$ represents eight strings with a 1 in the first position and a 0 in the second position. From Table 1.1, we observe that there is only one string representing this schema H_1 in the initial population and that there are two strings representing this schema in the new population. On the other hand, even though there was one representative string of the schema $H_2 = (0 \ 0 \ * \ * \ *)$ in the initial population, there is not one in the new population. There are a number of other schemata that we may investigate and conclude whether the number of strings they represent is increased from the initial population to the new population or not.

The so-called schema theorem provides an estimate of the growth of a schema H under the action of one cycle of the above tripartite GA. Holland (1975) and later Goldberg (1989) calculated the growth of the schema under a selection operator and then calculated the survival probability of the schema under crossover and mutation operators, but did not calculate the probability of constructing a schema from recombination and mutation operations in a generic sense. For a single-point crossover operator with a probability p_c, a mutation operator with a probability p_m, and the proportionate selection operator, Goldberg (1989) calculated the following lower bound on the schema growth in one iteration of a GA:

$$m(H, t+1) \geq m(H, t) \frac{f(H)}{f_{avg}} \left[1 - p_c \frac{\delta(H)}{\ell - 1} - p_m o(H)\right], \tag{1.5}$$

where $m(H, t)$ is the number of copies of the schema H in the population at generation t, $f(H)$ is the fitness of the schema (defined as the average fitness of all strings representing the schema in the population), and f_{avg} is the average fitness of the population. The above inequality leads to the schema theorem (Holland, 1975), as follows.

> Short, low-order, and above-average schemata receive exponentially increasing number of trials in subsequent generations.

A schema represents a number of similar strings. Thus, a schema can be thought of as representing a certain region in the search space. For the above function, the schema $H_1 = (1 \ 0 \ * \ * \ *)$ represents strings with x values varying from 1.621 to 2.330 with function values varying from 0.999 to 0.725. On the other hand, the schema $H_2 = (0 \ 0 \ * \ * \ *)$ represents strings with x values varying from 0.0 to 0.709 with function values varying from 0.0 to 0.651. Since our objective is to maximize the function, we would like to have more copies of strings representing schema H_1 than H_2. This is what we have accomplished in Table 1.1 without having to count all of these competing schema and without the knowledge of the complete search space, but by manipulating only a few instances of the search space. Let us use the inequality shown in equation (1.5) to estimate the growth of H_1 and H_2. We observe that there is only one string (the second string) representing this schema, or $m(H_1, 0) = 1$. Since all strings are used in the crossover operation and no mutation is used, $p_c = 1.0$ and $p_m = 0$. For the schema H_1, the fitness $f(H_1) = 0.898$, the order $o(H_1) = 2$, and the defining length $\delta(H_1) = 1$. In addition, the average fitness of the population is $f_{avg} = 0.569$. Thus, we obtain from equation (1.5):

$$\begin{aligned} m(H_1, 1) &\geq (1) \cdot \frac{0.898}{0.569} \left[1 - (1.0)\frac{1}{5-1} - (0.0)(2)\right], \\ &= 1.184. \end{aligned}$$

The above calculation suggests that the number of strings representing the schema H_1 must increase. We have two representations (the second and third strings) of this schema in the next generation. For the

schema H_2, the estimated number of copies using equation (1.5) is $m(H_2, 1) \geq 0.133$. Table 1.1 shows that no representative string of this schema exists in the new population.

The schema H_1 for the above example has only two defined positions (the first two bits) and both defined bits are tightly spaced (very close to each other) and contain the possible near-optimal solution (the string `(1 0 0 0 0)` is the optimal string in this problem). The schemata that are short, low-order, and above-average are known as the *building blocks*. While GA operators are applied on a population of strings, a number of such building blocks in various parts along the string get emphasized, such as H_1 (which has the first two bits in common with the true optimal string) in the above example. Note that although H_2 is short and low-order, it is not an above-average schema. Thus, H_2 is not a building block. This is how GAs can emphasize different short, low-order and above-average schemata in the population. Once adequate number of such building blocks are present in a GA population, they get combined together due to the action of the GA operators to form bigger and better building blocks. This process finally leads a GA to find the optimal solution. This hypothesis is known as the *Building Block Hypothesis* (Goldberg, 1989).

1.4.4 Constraint Handling

As outlined elsewhere (Michalewicz and Schoenauer, 1996), most constraint handling methods which exist in the GA literature can be classified into five categories, as follows:

1. Methods based on preserving feasibility of solutions.
2. Methods based on penalty functions.
3. Methods biasing feasible over infeasible solutions.
4. Methods based on decoders.
5. Hybrid methods.

However, in most applications, the penalty function method has been used with GAs. Usually, an exterior penalty term (Deb, 1995, Reklaitis at el., 1983), which penalizes infeasible solutions, is preferred. Based on the constraint violation $g_j(\mathbf{x})$ or $h_k(\mathbf{x})$, a bracket-operator penalty term is added to the objective function and a penalized function is formed:

$$F(\mathbf{x}) = f(\mathbf{x}) + \sum_{j=1}^{J} R_j \langle g_j(\mathbf{x}) \rangle + \sum_{k=1}^{K} r_k |h_k(\mathbf{x})|, \tag{1.6}$$

where R_j and r_k are user-defined penalty parameters. The bracket-operator $\langle\ \rangle$ denotes the absolute value of the operand, if the operand is negative. Otherwise, if the operand is non-negative, it returns a value of zero. Since different constraints may take different orders of magnitude, it is essential to normalize all constraints before using the above equation. A constraint $\underline{g}_j(\mathbf{x}) \geq b_j$ can be normalized by using the following transformation:

$$g_j(\mathbf{x}) \equiv \underline{g}_j(\mathbf{x})/b_j - 1 \geq 0.$$

Equality constraints can also be normalized similarly. Normalizing constraints in the above manner has an additional advantage. Since all normalized constraint violations take more or less the same order of magnitude, they all can be simply added as the overall constraint violation and thus only one penalty parameter R will be needed to make the overall constraint violation of the same order as the objective function:

$$F(\mathbf{x}) = f(\mathbf{x}) + R \left[\sum_{j=1}^{J} \langle g_j(\mathbf{x}) \rangle + \sum_{k=1}^{K} |h_k(\mathbf{x})| \right]. \tag{1.7}$$

When an EA uses a fixed value of R in the entire run, the method is called the *static* penalty method. There are two difficulties associated with this static penalty function approach:

1. The optimal solution of $F(\mathbf{x})$ depends on penalty parameters R. Users usually have to try different values of R to find which value would steer the search towards the feasible region. This requires extensive experimentation to find any reasonable solution. This problem is so severe that some researchers have used different values of R depending on the level of constraint violation (Homaifar et al., 1994),

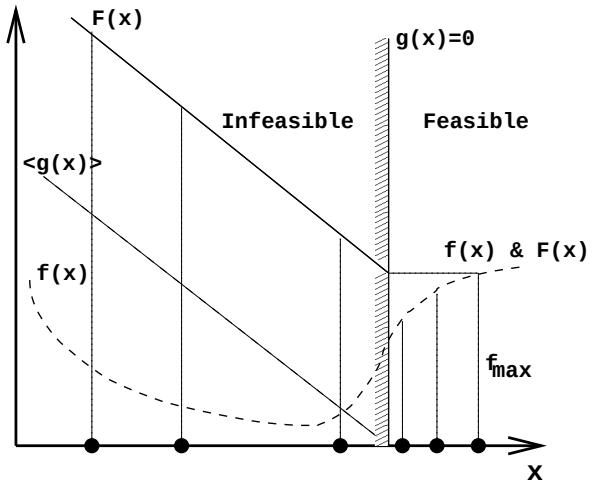

Figure 1.2: A constraint handling strategy without any penalty parameter.

while some have used a sophisticated temperature-based evolution of penalty parameters through generations (Michalewicz and Attia, 1994) involving a few parameters describing the rate of evolution. We will discuss these *dynamically* changing penalty methods a little later.

2. The inclusion of the penalty term *distorts* the objective function (Deb, 1995). For small values of R, the distortion is small, but the optimum of $F(\mathbf{x})$ may not be near the true constrained optimum. On the other hand, if a large R is used, the optimum of $F(\mathbf{x})$ is closer to the true constrained optimum, but the distortion may be so severe that $F(\mathbf{x})$ may have artificial locally optimal solutions. This primarily happens due to interactions among multiple constraints. EAs are not free from the distortion effect caused due to the addition of the penalty term in the objective function. However, EAs are comparatively less sensitive to distorted function landscapes due to the stochasticity in their operators.

A recent study (Deb, 2000) suggested a modification, which eliminates both the above difficulties by *not* requiring any penalty parameter:

$$F(\mathbf{x}) = \begin{cases} f(\mathbf{x}), & \text{if } \mathbf{x} \text{ is feasible;} \\ f_{\max} + \sum_{j=1}^{J} \langle g_j(\mathbf{x}) \rangle + \sum_{k=1}^{K} |h_k(\mathbf{x})|, & \text{otherwise.} \end{cases} \tag{1.8}$$

Here, $f_{\max}$ is the objective function value of the worst feasible solution in the population. Figure 1.2 shows the construction procedure of $F(x)$ from $f(x)$ and $g(x)$ for a single-variable objective function. One fundamental difference between this approach and the previous approach is that the objective function value is *not* computed for any infeasible solution. Since all feasible solutions have zero constraint violation and all infeasible solutions are evaluated according to their constraint violations only, both the objective function value and constraint violation are not combined in any solution in the population. Thus, there is no need to have any penalty parameter R for this approach. Moreover, the approach is also quite pragmatic. Since infeasible solutions are not to be recommended for use, there is no real reason for one to find the objective function value for an infeasible solution. The method uses a binary tournament selection operator, where two solutions are compared at a time, and the following scenarios are always assured:

1. Any feasible solution is preferred to any infeasible solution.
2. Among two feasible solutions, the one having a better objective function value is preferred.
3. Among two infeasible solutions, the one having a smaller constraint violation is preferred.

To steer the search towards the feasible region and then towards the optimal solution, the method recommends the use of a niched tournament selection (where two solutions are compared in a tournament only if their Euclidean distance is within a pre-specified limit). This ensures that even if a few isolated solutions are found in the feasible space, they will be propagated from one generation to another for maintaining diversity among the feasible solutions.

1.5 Theoretical Modeling of Genetic Algorithms

Over the last decade, genetic algorithms have been applied to a numerous search and optimization problems, including various disciplines of sciences, engineering, and commerce. Although the success of these applications in various disciplines is a testimony of the working of GAs and demonstrates the broad-based applicability of GAs, the theoretical understanding of the working principles of GAs is a very recent phenomenon. This is mainly due to the stochasticity, multi-dimensionality, and multi-facetedness of GAs which make the theoretical developments difficult to achieve. In this section, we outline different techniques which have been used to attempt to have a better understanding of the working principles of a GA.

1.5.1 Functional Decomposition Approach

In this approach, simple models have been proposed to analyze one particular aspect of a GA. Instead of attempting to model all GA operators in unison, the operators have been classified into various classes based on their functionality in the search process. Such functional decomposition methods have been routinely used in engineering analysis of complex systems. Wright Brothers' functional decomposition approach of studying lift and drag aspects separately made them the first to discover the theory of flight. While lift issues were studied, drag was not considered and vice versa. The important and difficult matter of this approach is to determine a way to decompose different aspects of a complex system *functionally*. Although this approach is an approximate one, but usually provide useful insights of the working of a complex system.

In the case of GAs, David Goldberg and his students have been mainly engaged in using this approach for unfolding complex working principles of GAs (Goldberg, 1993) and in that process have suggested important guidelines for a successful GA simulation. In the following, we describe a few of such studies in brief.

Population Sizing

In a genetic algorithm, a decision about whether to have a '1' or a '0' has to be made at every bit position. The mechanism of using selection, crossover and mutation is one of the many ways of arriving at a decision. Holland viewed this decision-making in each bit independently and compared the process to a two-armed bandit game-playing problem. With one arm being fixed to either a 1 or a 0, the task is to find the arm which makes the maximum payoff (or fitness). With respect to the above definition of schema, every one-bit information is an order-one schema. Since the rest $(\ell - 1)$ bits are not considered, the fitness of one instantiation of any order-one schema is dependent on the exact value of bits at other locations.

In order to demonstrate the significance of an adequate population size in a GA, let us make use of a simple bimodal single-variable objective function for maximization:

$$f(x) = c_1 N(x, a_1, b_1) + c_2 N(x, a_2, b_2), \tag{1.9}$$

where the function $N(x, a_i, b_i)$ is a Gaussian function with a mean at a_i and a standard deviation of b_i. The search space is spanned over $[0, 1]$. By varying these six parameters, one can obtain functions of differing complexity. Let us choose the location of two maxima: $a_1 = 0.25$, and $a_2 = 0.80$. First, we use the following setting:

$$b_1 = 0.05, \quad b_2 = 0.05, \quad c_1 = 0.5, \quad c_2 = 1.0.$$

This makes the second maximum (at $x = 0.80$) the global maximum solution. Figure 1.3 shows the function. Let us consider two competing schemata $(H_0 \equiv 0 * \ldots *)$ and $(H_1 \equiv 1 * \ldots *)$ and analyze their growth under genetic operators. Figure 1.3 also shows that the left region $(x \in [0, 0.5])$ is represented by H_0 and the right region $(x \in [0.5, 1.0])$ is represented by the schema H_1. The average fitness of these two schemata are also shown with dashed lines. It is clear that the schema H_1 has a better fitness than H_0. Since the schema order and defining length are identical for both these schemata and H_1 is above-average, it becomes a building block. Thus, it is expected that a GA will make the right decision of choosing more solutions of H_1 than H_0. Since a GA can decide the correct direction of search by processing order-one schemata alone, a population of size two is essentially enough. However, note that the average fitness values shown by dashed lines in Figure 1.3 for any of the schemata do not adequately represent the fitness of all the strings which are represented by the schema. There is a large variation in the fitness values among the strings representing a schema. Because of this variability in fitness values, an appropriate number of copies in each region must be present to adequately represent the fitness variations. If we assume that λ (> 1) number of solutions are needed to statistically represent a region, an overall population of size 2λ is enough to solve the above problem by using a binary-coded GA.

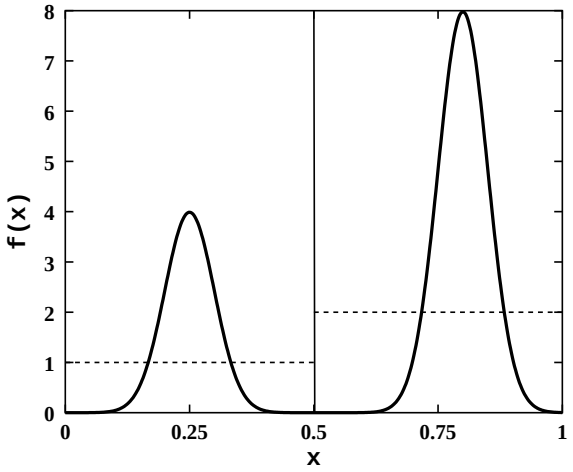

Figure 1.3: The simple bimodal function and average fitness values of two order-one schemata.

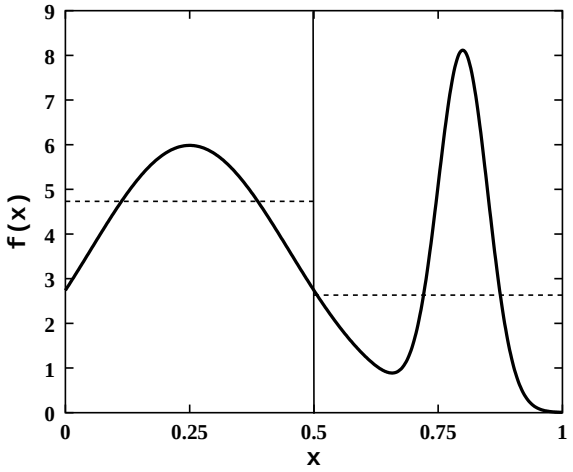

Figure 1.4: The modified bimodal function and average fitness values of two order-one schemata.

Table 1.2: Schema fitness values for the bimodal problem.

Schema fitness					
Order-one		Order-two		Order-three	
(0.000–0.500)	4.732	(0.000–0.250)	4.732	(0.000–0.125)	3.848
(0.500–1.000)	2.633	(0.250–0.500)	4.732	(0.125–0.250)	5.616
		(0.500–0.750)	1.828	(0.250–0.375)	5.616
		(0.750–1.000)	3.439	(0.375–0.500)	3.848
				(0.500–0.625)	1.808
				(0.625–0.750)	1.848
				(0.750–0.875)	6.324
				(0.875–1.000)	0.553

Now let us change the problem by choosing another parameter setting:

$$b_1 = 0.20, \quad b_2 = 0.05, \quad c_1 = 3, \quad c_2 = 1.$$

Figure 1.4 shows the function having two maxima. Now, the local maximum solution occupies a larger basin of attraction than the global maximum solution. When the fitnesses of H_0 and H_1 are computed (Table 1.2), it is observed that H_0 has a better fitness than H_1. Thus, if a population size of 2λ is used to solve the modified problem, a GA will be misled and emphasize more solutions of H_0 in a random initial population. Although this does not mean that a GA will not be able to recover from such early mistakes, a reliable application of the GA would be to make sure that it is started in the correct direction from the very first generation. In order to investigate the higher-order schema competitions, we have computed the fitness of four order-two competing schemata in Table 1.2, where the order of H_{00}, H_{01}, H_{10} and H_{11} is vertically downwards. These values are also shown pictorially in Figure 1.5. Here also, we observe that the schema (representing the region $0.750 \leq x \leq 0.875$) containing the global maximum solution does not win the competition. When we compute the order-three schema competitions (see Table 1.2), we observe that the schema containing the global maximum wins the competition. Figure 1.6 clearly shows that the schema containing the global maximum has the maximum fitness value. However, the overall function is favored for the local maximum. In order to recognize the importance of the global maximum solution, a schema with a narrower region must be emphasized. In fact, any higher-order (> 3) schema competition will favor the global maximum solution. Thus, if a GA has to find the global maximum in a reliable way, the GA has to start comparing order-three schemata with the variability in string fitness in mind. Any competition lower than order three will favor the local maximum solution. One of the ways to ensure that a GA will start processing from order-three schemata is to have a population large enough to house strings representing all 2^3 competing schemata. Let us recall that the former bimodal problem (see Figure 1.3) required us to

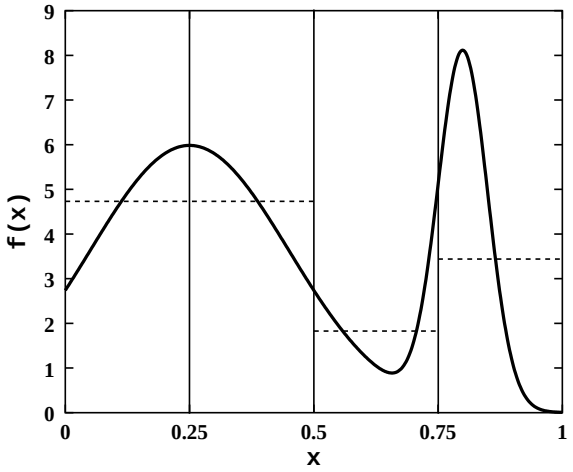

Figure 1.5: Average fitness values of different order-two schemata shown for the modified bimodal function.

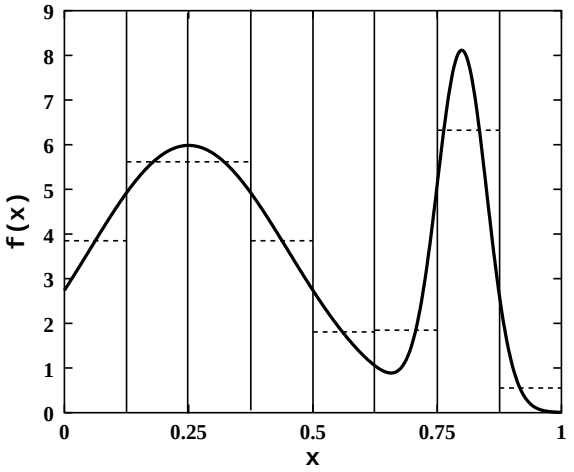

Figure 1.6: Average fitness values of different order-three schemata shown for the modified bimodal function.

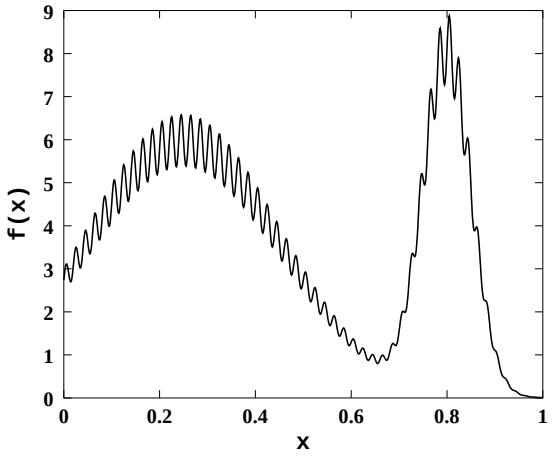

Figure 1.7: A function with a large variability in function values demands a large population size to find the global optimum.

start processing from order-one schemata (having 2^1 or two competing schemata) in order to proceed in the correct direction. Since here, eight schemata are competing, a population of size 8λ would be adequate to represent a sufficient number of strings from each of the eight competing schemata. Thus, the modified bimodal problem needs four times more population size in order to reliably find the global maximum solution compared to the original bimodal problem.

Thus, we observe that the population size is related to the complexity of the problem. The parameter λ depends on the variability in fitness values of strings representing a schema. For example, when keeping the location of two maxima, if the function is not as smooth (Figure 1.7), the required sample size λ needed to detect a *signal* from the *noise* would be higher. Based on these considerations, Goldberg et al. (1992) calculated a population sizing estimate for a binary-coded GA. Following this study, Harik et al. (1993) tightened the sizing expression by considering the schema competition from a gambler's ruin model.

Exploitation Versus Exploration

Besides choosing an appropriate population size, another important matter is the balance between exploitation caused by the selection operator and the exploration introduced by the chosen recombination and mutation operators. If the selection operator uses too much selection pressure, meaning that it emphasizes

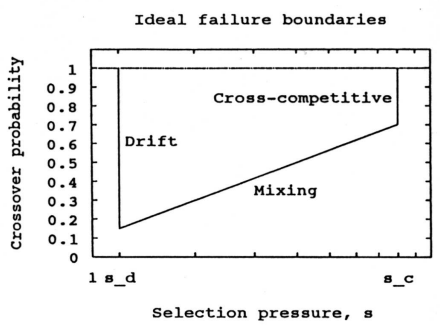

Figure 1.8: A theoretical control map for the working of a GA on the one-max problem (Goldberg et al., 1993).

the population-best solution too much by assigning many copies of it, the population loses its diversity very quickly. In order to bring back the diversity, the exploration power of the recombination and mutation operators must be large, meaning that these operators must be able to create solutions which are fairly different from the parent solutions. Otherwise, the population can become the victim of excessive selection pressure and eventually converge to a sub-optimal solution. On the other hand, if the selection pressure is very low, meaning that not much emphasis is given to the population-best solutions, the GA's search procedure behaves like a random search process. Although a qualitative argument of the balance between these two issues can be made and understood, a quantitative relationship between them is difficult to achieve.

Goldberg et al. (1992) and Thierens and Goldberg (1993) balanced the extent of exploitation and exploration issues by calculating the characteristic times of a selection and a crossover operator. In their earlier work (Goldberg and Deb, 1992), the *take-over times* t_s of a number of selection operators were calculated. The take-over time was defined as the number of generations required for the population-best solution to occupy all but one of the population slots by repetitive application of the selection operator alone. This characteristic time of a selection operator provides information about the speed with which the best solution in a population is emphasized. It was observed that binary tournament selection and linear ranking selection (with an assignment of two copies to the best solution) have the same take-over times. Proportionate selection is much slower than tournament selection.

For the uniform crossover operator, investigators have calculated the *mixing time* t_c, which refers to the number of generations required before repetitive application of the crossover operator alone can find a desired solution. This time gives information about how long a GA would have to wait before an adequate mixing of population members can produce the desired solutions.

By comparing the order of magnitudes of these two characteristic times, investigators argued that a GA will work successfully if the following relationship holds:

$$p_c \geq A \ln s, \tag{1.10}$$

where A is a constant which relates to the string length and population size and s is the selection pressure. On the one-max test problem, where the objective is to maximize the number of 1s in a string, Figure 1.8 shows the theoretical *control map* for successful GAs. A similar control map is obtained using GA simulation results (Figure 1.9). What is interesting to note from these figures is that a GA with any arbitrary parameter setting is not expected to work well even on a simple problem. A GA with a selection pressure s and a crossover probability p_c falling inside the control map finds the desired optimum. The above problem is a bit-wise linear problem, where a decision can be made in each bit, independent of the decisions taken at other bits. Even for this problem, a tournament selection with a large tournament size (such as $s = 20$ or so) and the uniform crossover operator with a small crossover probability (such as $p_c = 0.2$ or so) and no mutation, is not expected to find the true optimum solution. Although these parameter settings are probably extreme ones to choose in any application, they provide an understanding of the importance of interactions among GA operators and their parameter settings in performing a successful GA run.

For a selection pressure lower than a critical value, GAs can drift to any arbitrary solution and for a very large selection pressure, GAs cause important building blocks to compete among themselves, thereby not finding the desired optimum. These two cases become the two extreme bounds on the selection pressure in the above control map.

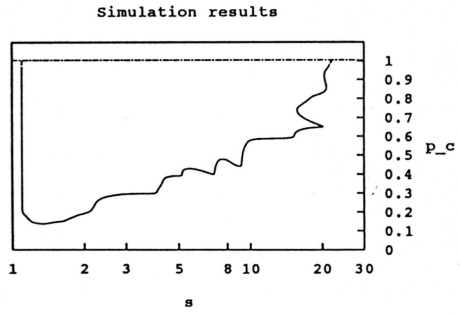

Figure 1.9: Simulation results show the working region for a GA on the one-max problem (Goldberg et al., 1993).

GA Operator Interactions Under Fixed Number of Trials

It is clear from the description of the working principles of a GA that GA operators are tunable with the parameters associated with each of them. Since there exist flexibilities for changing the importance of one operator over another or flexibilities in using a different representation scheme to suit a problem, GAs are widely applicable to various types of problems. As we have seen in the previous section, along with this flexibility a burden is placed on the part of the user to choose an appropriate combination of GA operators and their parameters. While facing a real-world problem, the overall time to solve the problem is often fixed. This means that whatever algorithm is used to solve the problem at hand, we are only allowed to use a fixed number of trials (or function evaluations). For the sake of our discussion here, let us say that we are allowed to use a total of S trials to solve a problem. Since in a generational GA we create N new offspring at every generation, this means that we are allowed to run a GA having a population size N for a maximum of $t_{\max} = S/N$ generations. If N is small, a large number of generations are allowed. On the other hand, if N is large, a small number of generations will be allowed. Then, an important point to ponder is what population size is appropriate to choose when the overall number of trials is fixed. A related question to ask is: 'Given the adequate population size, what combination of GA operators is appropriate?' It is important to realize that the answer to these questions will be different if the maximum number of allowed trials does not have a bound. Several researchers have attempted to find such GA operators and their parameter interactions for fixed number of trials, simply because such considerations are pragmatic in solving real-world problems.

Deb and Agrawal (1998) conducted a series of experiments with different GA operators and parameter settings and applied these to different problems of varying difficulties. The outcome of this study is important and is outlined in the following:

1. For simpler problems (such as well-behaved unimodal or linear problems), a GA with a selection operator and a crossover or a mutation operator, or a combination of crossover and mutation operators, can all work satisfactorily. However, there is a distinct difference in the required population size in each case. For a selecto-mutation GA, a small population size (such as 3–6) provides the optimum performance. Since a selecto-mutation GA is similar to a local search approach, instead of more population members it requires a larger number of iterations to navigate its search towards the optimum. On the other hand, for a selecto-recombinative GA (with no mutation operator), the population size requirement is rather high. A GA with a crossover requires an adequate population size (described in Section 1.5.1) to steer the search in the right direction. Since the search relies on combining salient building blocks, the building block discovery and their emphasis requires an adequate number of population members. However, once the salient building blocks are found, not many generations are needed to combine them together. Thus, although a larger population size is in order, the number of generations required may be comparatively smaller. This is in contradiction to the common belief that since GAs use a population of solutions in each iteration, they are computationally more expensive than a local search algorithm.

2. For difficult problems (where difficulty can come from multi-modality, dimensionality of the search space, ruggedness of the fitness function, etc.), selecto-mutation GAs do not work successfully in finding the correct optimum solution. However, selecto-recombinative GAs can find the correct optimum with an adequate population size. Oates et al. (1999) observed similar results on a number of other problems.

1.5.2 Statistical Dynamics Approach

On a more rigorous note, GAs have been mainly analyzed from two aspects. On the macroscopic aspect, the dynamics of GA population statistics, such as population mean fitness, population variance in fitness, etc. are estimated. We describe some of these procedures in this subsection. On the microscopic aspect, the dynamics of each individual string in a population is estimated. For obvious reasons, this latter approach is more detailed and we defer its discussion till the next subsection.

To illustrate the statistical dynamics approach, let us consider a simple function of boolean variables (largely known as the 'onemax' problem), in which the objective is to maximize the number of 1s in a string of length ℓ. Here, we consider $\ell = 100$. Figure 1.10 shows the simulation results of a GA with a binary tournament selection and bit-wise mutation ($p_m = 0.01$). The histogram of population fitness

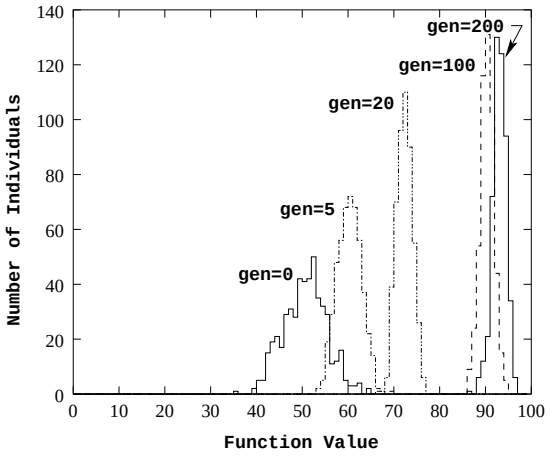

Figure 1.10: Histogram of individual fitness values at different generations for the onemax problem.

values at various generations of a single run are shown in the figure. In the initial population (marked as generation=0), it is expected to have 50% 1s in any string, thereby showing a distribution of fitness values with a mean close to 50. However, with generation, the mean population fitness increases and approaches the maximum fitness of 100. There are a several important matters to notice from such a plot.

1. The variance in the population fitness reduces with generation number. This implies that the strings in a population get more similar with generation number.

2. The rate of increase in the population mean fitness slows down as it approaches the optimum. This is evident by observing the progress in the first 100 generations of the run and that in the subsequent 100 generations of the same run.

Although the above figure portrays the population details somewhat in each generation, the distribution of fitness values in a population can also be captured by means of some statistics involving the fitness distribution. For example, the mean population fitness and the standard deviation in population fitness values can provide some idea of the distribution of fitness as it evolves with generation. Figures 1.11 and 1.12 show these quantities by solid lines. It is clear that while the mean population fitness is approaching the optimum, the standard deviation of population fitness reduces.

It is clear from the description of a GA that it is stochastic in the sense that it begins its search from a random population, its selection operator involves sampling and its crossover and mutation operators probabilistically change strings. Thus, it is expected that the dynamics of a GA will vary from one simulation run to another. For example, if we keep a history of the population mean fitness as it evolves from generation to generation, two independent runs of a GA with identical GA parameters such as population size, GA operators etc. will not produce exactly the same statistics. In the statistical dynamics approach, researchers are interested in finding the dynamics of ensemble average of some population statistics such as population mean fitness, variance in population fitness, skewness in population fitness etc. with generation. Figures 1.11 and 1.12 show the ensemble averages of mean and standard deviation in population fitness values of 100

different GA simulations (with $p_m = 0.01$), each starting from a different set of random solutions. In this

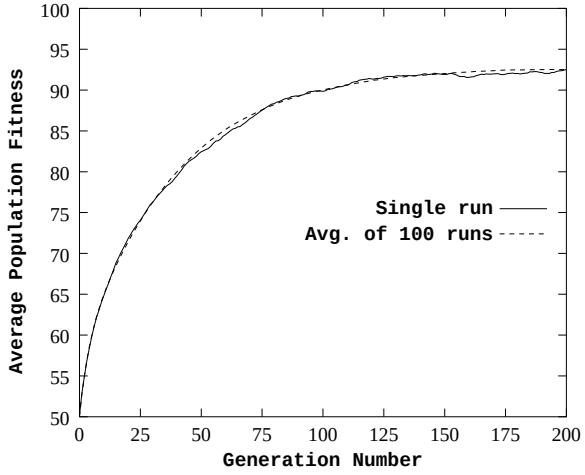

Figure 1.11: Mean population fitness with generation number for the onemax problem.

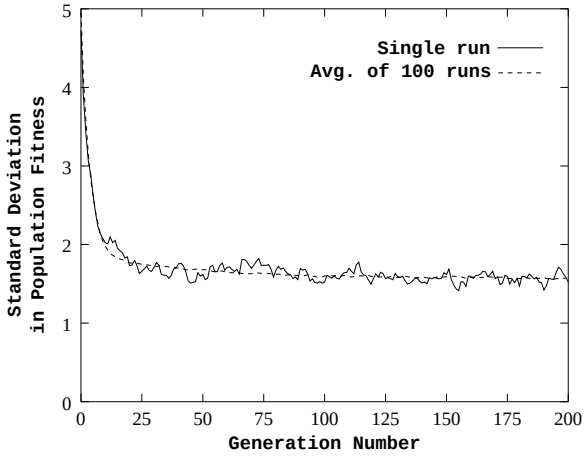

Figure 1.12: Standard deviation in population fitness with generation number for the onemax problem.

problem with the chosen GA parameters, the figures show that the difference between one simulation run and the average of 100 simulation runs is not significant. Figure 1.13 shows the corresponding histogram for the ensemble average of fitness values. It is clear from these figures that the variation of ensemble average values is more gradual than that of a single simulation run. It is this fact that has motivated a group of researchers to attempt to find theoretical models for predicting the dynamics of ensemble average of population statistics. In the following, we describe one such results from Prügel-Bennett and Rogers (2001).

Modeling of Ensemble Population Statistics

In such a model, the first task is to identify which population statistics are to be studied. Most studies so far have considered the ensemble mean $\overline{\mu}$, the ensemble standard deviation $\overline{\sigma}$, the ensemble skewness $\overline{\kappa_2}$, and the ensemble kurtosis $\overline{\kappa_3}$. Starting from initial values of these statistics (assuming a random initial population), the difference equations of these quantities are constructed under individual genetic operations, as follows:

$$\begin{pmatrix} \overline{\mu}(t) \\ \overline{\sigma}(t) \\ \overline{\kappa_2}(t) \\ \overline{\kappa_3}(t) \end{pmatrix} \underset{\longrightarrow}{\text{Selection}} \begin{pmatrix} \overline{\mu}^{(s)}(t) \\ \overline{\sigma}^{(s)}(t) \\ \overline{\kappa_2}^{(s)}(t) \\ \overline{\kappa_3}^{(s)}(t) \end{pmatrix} \underset{\longrightarrow}{\text{Crossover}} \begin{pmatrix} \overline{\mu}^{(c)}(t) \\ \overline{\sigma}^{(c)}(t) \\ \overline{\kappa_2}^{(c)}(t) \\ \overline{\kappa_3}^{(c)}(t) \end{pmatrix}$$

$$\underset{\longrightarrow}{\text{Mutation}} \begin{pmatrix} \overline{\mu}^{(m)}(t) \\ \overline{\sigma}^{(m)}(t) \\ \overline{\kappa_2}^{(m)}(t) \\ \overline{\kappa_3}^{(m)}(t) \end{pmatrix} = \begin{pmatrix} \overline{\mu}(t+1) \\ \overline{\sigma}(t+1) \\ \overline{\kappa_2}(t+1) \\ \overline{\kappa_3}(t+1) \end{pmatrix}.$$

For the onemax problem, Prügel-Bennett and Rogers (2001) have calculated the following difference equations assuming an infinite population and that the fitness distributions have only non-zero mean and variance. However, they calculated the evolution of the skewness term as well for the onemax function. For simplicity, we only present the results for the mean and variance terms. Under the tournament selection, the mean and variance terms are as follows:

$$\mu^{(s)}(t) = \mu(t) + \frac{\sigma(t)}{\sqrt{\pi}}, \tag{1.11}$$

$$\sigma^{(s)^2}(t) = \left(1 - \frac{1}{\pi}\right)\sigma^2(t). \tag{1.12}$$

Under the mutation operator, the quantities change to the following:

$$\mu^{(m)}(t) = p_m \ell + (1 - 2p_m)\mu^{(s)}(t), \tag{1.13}$$

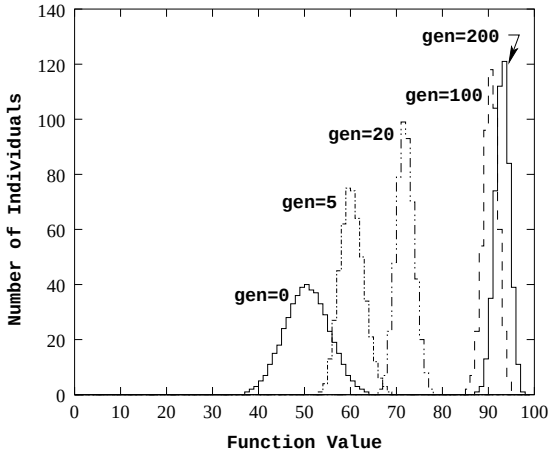

Figure 1.13: Histogram of ensemble average of individual fitness values of 100 simulation runs at different generations for the onemax problem.

$$\sigma^{(m)^2}(t) = p_m(1-p_m)\ell + (1-2p_m)^2\sigma^{(s)^2}(t). \tag{1.14}$$

Substituting the terms for $\mu^{(s)}(t)$ and $\sigma^{(s)^2}(t)$ in the above equations, we have the mean fitness and variance of the population fitness after one iteration of a selecto-mutation GA:

$$\mu(t+1) = p_m\ell + (1-2p_m)\mu(t) + \frac{(1-2p_m)}{\sqrt{\pi}}\sigma(t), \tag{1.15}$$

$$\sigma^2(t+1) = p_m(1-p_m)\ell + (1-2p_m)^2\left(1-\frac{1}{\pi}\right)\sigma^2(t). \tag{1.16}$$

The above two difference equations will be started with $\mu(0) = \ell/2$ and $\sigma^2(0) = \ell/4$. Although an exact solution to these equations may be difficult to achieve, they represent an approximate model of the genetic operations. Such systems of equations can be iterated to get an insight to the working of a GA. Figures 1.14 and 1.15 shows the outcome of the iteration of the above equations using $p_m = 0.01$. Comparing with

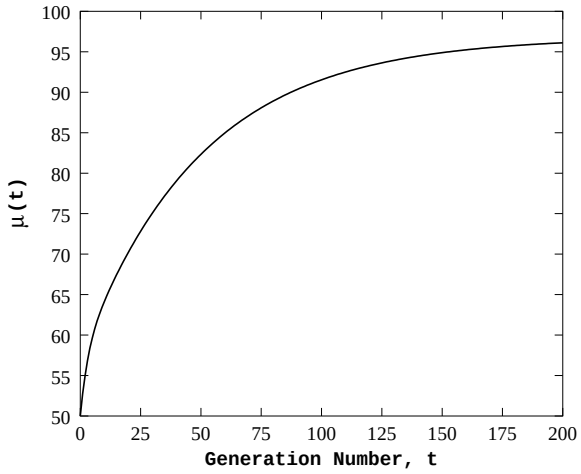

Figure 1.14: Mean population fitness with generation number for the onemax problem obtained by theory.

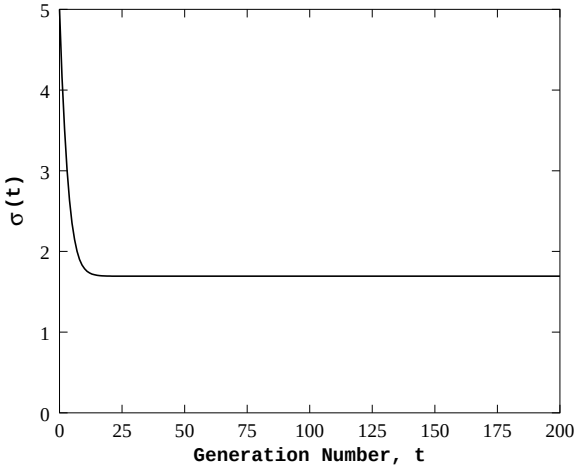

Figure 1.15: Standard deviation in population fitness with generation number for the onemax problem obtained by theory.

simulated results shown in Figures1.11 and 1.12, it can be seen that that theoretical results predict the GA

performance quite well. Such equations can also be used to find the attractors and to establish the stability of GAs at the attractors. For the above system of equations, the attractors are as follows:

$$\mu^* = \ell/2 + \frac{1-2p_m}{2p_m\sqrt{\pi}}\sigma^*, \tag{1.17}$$

$$\sigma^* = \sqrt{\frac{p_m(1-p_m)\ell}{1-(1-2p_m)^2(1-1/\pi)}}. \tag{1.18}$$

For a very small mutation rate ($p_m \ll 1$), the final mean is $\mu^* = (\ell + \sqrt{\ell/p_m})/2$ (for $p_m \geq 1/\ell$) and the variance is $\sigma^{*2} = \pi p_m \ell$. A careful analysis of these equations will also reveal that there are two phases of operation. In the initial transient phase, the variance of the population decays very fast. In the next long phase, the mean approaches the optima slowly with a small reduction rate in the variance.

In addition to correcting the above terms by including more population statistics, researchers have also been able to include crossover operations into the calculation for certain fitness functions. For the onemax problem, Prügel-Bennett and Rogers (2001) have shown that the presence of a uniform crossover delays the first phase of loosing diversity, thereby allowing a better search.

Researchers have also computed the above quantities for a finite population. In such an analysis, first an infinite population model is achieved. Thereafter, the equations are corrected by probabilistically choosing a sample of size N from the infinite population model. For an ensemble of populations, this will not affect the mean population fitness, but the variance of population fitness for a finite population becomes smaller than that in the infinite population model. In other words, GA with a finite population looses its variability quickly and may lead to a poor performance.

One nice aspect of this approach, however, is that it can be used to study parametric interactions. Attractors and their stability can be understood using the well-established theories of non-linear dynamics (Shapiro, 2001). However, in order to use these tools, first the user needs to develop the difference equation models for the GA operators at hand. It is clear that the effect of the selection operator depends on the chosen objective function. Moreover, since information about individual solution in a population is not processed independently, the evolution of the best solution in a population is difficult to track using this approach.

1.5.3 Dynamical Systems Model

In this model, a population p is defined as a point in the space of all possible populations Λ. The effect of one generation of a GA is to change the population to another population in Λ. In these studies the emphasis is then to find the expected next population from the current population. Continuing this process will lead to finding the attractor populations. This theory is largely due to Micheal D. Vose and details of the analysis procedure exist in his recently published seminal book (Vose, 1999).

For an ℓ-bit binary coding, there are $s = 2^\ell$ strings. Any population p containing N strings can be represented as a s-dimensional real vector with an element $p_i = a_i/N$, where a_i is the number of string i in the population. For example, with a 2-bit string coding, a population having one copy of `00`, no copies of `01`, six copies of `10`, and three copies of `11`, the population vector is $p = (0.1, 0.0, 0.6, 0.3)^T$. From the above discussion, we observe three properties of a valid population vector:

1. Every component of a valid population vector is non-negative.
2. The sum of all components of a valid population vector is one.
3. For a finite population of size N, each component of a valid population vector will be an integer multiple of $1/N$.

The set of all population vectors (or points) which satisfy all of the above conditions is called a *simplex*. Thus, for an infinite population, any non-negative entry in a population vector is allowed provided that the sum of all entries is one. However, for a finite population, only a finite number of populations qualify for the simplex. For example, with $s = 2$, all populations lying on the straight line shown in Figure 1.16 is a member of the simplex in the case of an infinite population, whereas for $N = 10$, only the points marked by circles qualify for the simplex.

Given a population p, it is now important to find the expected population after the action of genetic operators. Under proportionate selection, each string i has a probability $f(i)/(N\bar{f}(p))$ to be selected. Thus,

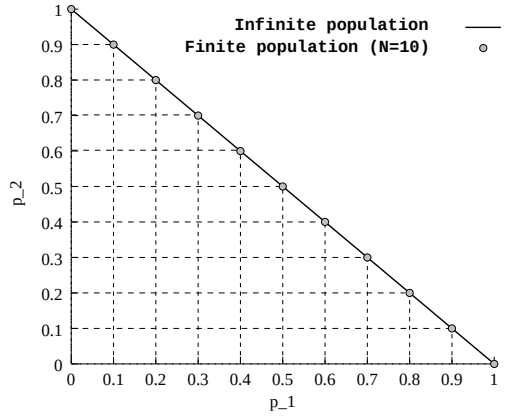

Figure 1.16: Simplex for the infinite and finite population ($N = 10$) GAs.

the expected population vector is

$$p^{(s)} = \frac{1}{\bar{f}(p)} Dp, \tag{1.19}$$

where D is a diagonal matrix with $D_{i,i} = f(i)$. The above equation can be rewritten as

$$Dp = \bar{f}(p)p^{(s)}.$$

Vose (1999) has also considered the effect of crossover and mutation on population $p^{(s)}$. These additional operators have an effect of multiplying the left term by two characteristic matrices ($\mathcal{C}$ and $\mathcal{M}$) containing the crossover and mutation parameters:

$$\mathcal{CM}Dp(t) = \bar{f}(p(t))p(t+1). \tag{1.20}$$

At the fixed-point, the above equation reduces to the following eigenvalue problem:

$$\mathcal{A}p = \lambda p. \tag{1.21}$$

Here $\mathcal{A} = \mathcal{CM}D$ and $\lambda = \bar{f}(p)$. For the details of the construction procedure of these matrices, readers are referred to the original study (Vose 1999) and Rowe (2001). The Perron-Frobenius theorem suggests that for a positive real entries in $\mathcal{A}$, there exists only one eigenvector which lie in the simplex Λ and this happens for the leading eigenvalue of the above system. Thus, the solution of the above equation will reveal the fixed-point population vector (the eigenvector corresponding to the leading eigenvalue). As a byproduct, the corresponding eigenvalue would represent the mean fitness of the fixed-point population.

For an infinite population, the eigenvector will directly represent the fixed-point. However, for finite populations, the eigenvector of the above equation may not be a feasible member of the simplex. For example, with $s = 4$, if the eigenvector of the above equation is $p^* = (0.2156, 0.1762, 0.5432,$
$0.0650)^T$, a population size of $N = 10$ cannot ever have this population vector in the simplex. The important question is then to find which member of the simplex is the attractor for a finite population? To answer this question, we have to calculate the transition probability of one population to change to another population under genetic operations.

For a given population $p(t)$, the probability distribution over all possible next populations, dictated by the equation $p(t+1) = \mathcal{G}(p(t))$ has been calculated by Vose and Liepins (1991). The probability that the next population is q, given that the current population is p, is

$$N! \prod_{i=0}^{s-1} \frac{(\mathcal{G}(p)_i)^{Nq_i}}{(Nq_i)!}.$$

For any population p, $\mathcal{G}(p)$ is the expected next population. The norm of the vector $(\mathcal{G}(p) - p)$ (or, $\|\mathcal{G}(p) - p\|$) will indicate a *distance* (or force) between the expected next population and the current population. Thus, it

can be argued that in the case of finite populations, although the eigenvector p^* is not a feasible population, GAs will spend more time at a population p for which the $\|p - p^*\|$ is smaller. For the above example case, a GA is expected to spend more time in a population $p = (0.2, 0.2, 0.5, 0.1)^T$ (with 20% strings are at `00` and `01`, 50% strings at `10`, and 10% strings are `11`) than any other population. This population p is then the likely attractor of the given problem.

Arguing from the distance point of view, the operator $\mathcal{G}$ can be considered continuous in the vicinity of the fixed point. This suggests an interesting phenomenon about the infinite population fixed points applied to finite populations. Since a feasible population vector is usually not the infinite population fixed point, feasible populations close to the infinite population fixed point will have a large probability for a GA to get attracted to. Although it has been argued that the simplex Λ contains exactly one fixed-point in the case of infinite population, in certain problems there could be more than one fixed points which lie just outside the simplex. For example, an infeasible fixed-point (the eigenvector corresponding to the second largest eigenvalue) could be $p^* = (0.5821, 0.3301, -0.0436, 0.1314)^T$. Although this infinite population fixed point is not feasible, GAs will have a large probability to get attracted to one of the neighboring feasible populations. In the above example, the feasible population where a GA may stay for a long time is $p = (0.6, 0.3, 0.0, 0.1)^T$. Such attractors are called *meta-stable* attractors.

The presence of mutation operators guarantees that any population can be visited from a population in a finite time (Davis and Principe, 1991). Thus, the presence of many such meta-stable attractors may cause a GA to wander from one attractor to another. With a small population size, the granularity of the simplex is less and many population vectors will act as meta-stable populations, thereby causing a GA to be attracted to a false optimum more often. However, with a large population size, population vectors with relatively smaller forces will qualify as attractors and will therefore spend more time in such meta-stable states than that with a smaller population size.

With a careful modeling of the GA at hand, this approach will provide interesting insights about different attractors and their stabilities as a function of population size. Since a complete population vector is evolved over generation, the best population member can also be tracked. However, since all 2^ℓ strings are involved with a matrix size of $2^\ell \times 2^\ell$, computer simulations for a reasonable-sized problem becomes a difficulty.

1.6 Advanced Genetic Algorithms

There exists a number of extensions to the simple GA described above. Interested readers may refer to the GA literature for details:

Real-coded GA: Variables taking real values are used directly, instead of using a binary coding of them. Although identical selection operators described here can be used, the trick lies in developing efficient crossover and mutation operators (Deb and Agrawal, 1995; Deb and Kumar, 1995; Eshelman and Schaffer, 1992; Kita et al., 1998; Tsutsui et al., 1999) Among these studies, some crossover operators, such as simulated binary crossover (SBX), blend crossover (BLX), fuzzy recombination operator (FR) use a variable-by-variable crossover, where an adaptive probability distribution using two parents is used to create one or two new offspring solutions. Some other crossover operators, such as UNDX and SPX use multinomial probability distribution among many parent solutions to create more than one offspring solutions. In most studies, a zero-mean normal distribution with an adaptive variance term is used as the mutation operator.

Since real numbers are used directly, the real-coded GAs eliminate the arbitrary precision and Hamming cliff problem associated with the binary GAs. Moreover, since adaptive probability distributions are used, some of these GAs have demonstrated their ability to self-adapt in non-stationary problems.

Micro GA: A small population size (of the order of 4 or 5) is used (Krishnakumar, 1989). This GA solely depends on the mutation operator, since such a small population cannot take advantage of the discovery of good partial solutions by a selecto-recombination GA. However, for unimodal and simple problems, micro-GAs are good candidates. For problems where the function evaluations are expensive, many researchers have used micro-GAs with a small population size in the expectation of finding a reasonable solution.

Knowledge-augmented GA: GA operators and/or the initial population is assisted with problem knowledge, if available. In most problems, some problem information is available and generic GA operators mentioned in this paper can be modified to make the search process faster (Davidor, 1991; Deb, 1993).

Hybrid GA: A classical greedy search operator is used starting from a solution obtained by a GA. Since a GA can find a good regions in the search space quickly, using a greedy approach from a solution in the global basin may make the overall search effort efficient (Powell and Skolnick, 1989; Kelly and Davis, 1991).

Multimodal GA: Due to the population approach, GAs can be used to find multiple optimal solutions in one simulation of a GA run. In such a multi-modal GA, only the reproduction operator needs to be modified. In one implementation, the raw fitness of a solution is degraded with its *niche count*, an estimate of the number of neighboring solutions. It has been shown that if the reproduction operator is performed with the degraded fitness values, stable subpopulations can be maintained at various optima of the objective function (Deb, 1989; Deb and Goldberg, 1989; Goldberg and Richardson, 1987). This allows GAs to find multiple optimal solutions simultaneously in one single simulation run.

Multi-objective GA: Most real-world search and optimization problems involve multiple conflicting objectives, of which the user is unable to establish a relative preference. Such considerations give rise to a set of multiple optimal solutions, largely known as the Pareto-optimal solutions or inferior solutions.

Multiple Pareto-optimal solutions are found simultaneously in a population. A GA is unique optimization algorithm in solving multi-objective optimization problems in this respect. In one implementation, non-domination concept is used with all objective functions to determine a fitness measure for each solution. Thereafter, the GA operators described here are used as usual. On a number of multi-objective optimization problems, this non-dominated sorting GA has been able to find multiple Pareto-optimal solutions in one single run (Deb, 2001; Fonseca and Fleming, 1993; Horn et al., 1994; Srinivas and Deb, 1994).

Non-stationary GA: The concept of diploidy and dominance can be implemented in a GA to solve non-stationary optimization problems. Information about earlier good solutions can be stored in recessive alleles and when needed can be expressed by suitable genetic operators (Goldberg and Smith, 1987).

Scheduling GA: Job-shop scheduling, time tabling, traveling salesman problems are solved using GAs. A solution in these problems is a permutation of N objects (name of machines or cities). Although reproduction operator similar to one described here can be used, the crossover and mutation operators must be different. These operators are designed in order to produce offsprings which are valid and yet have certain properties of both parents (Davis, 1991; Goldberg, 1989; Starkweather, 1991).

1.7 Conclusions

In this paper, we have described a new but potential search and optimization algorithm, originally conceived by John Holland about three decades ago, but now gained a lot of popularity. A genetic algorithm (GA) is different from other classical search and optimization methods in a number of ways: it does not use gradient information; it works with a set of solutions instead of one solution in each iteration; it works on a coding of solutions instead of solutions themselves; it is a stochastic search and optimization procedure; and it is highly parallelizable. GAs are finding increasing popularity primarily because of their wide spread applicability, global perspective, and inherent parallelism.

We have also discussed three different approaches for understanding theoretical aspects of GAs. Of them, the functional decomposition approach allows a simple yet approximate method of understanding GA parameter interactions. The statistical mechanics and dynamical systems approaches are sophisticated and difficult to pursue, but will provide a more theoretically sound analysis. All of these approaches have merits and demerits and must be used conveniently and appropriately, but certainly should not be used in a competitive sense. Only with these analysis tools at hand and the results derived from such analyses, GA users will be more confident and wise in choosing GA operators and parameters, a matter of which will take the whole effort of GA research and application for the past thirty years to its well-deserved promise land.

References

Bagley, J. D. (1967). The behavior of adaptive systems which employ genetic and correlation algorithms (Doctoral dissertation, University of Michigan). *Dissertation Abstracts International, 28*(12), 5106B. (University Microfilms No. 68-7556).

Bethke, A. D. (1981). Genetic algorithms as function optimizers (Doctoral dissertation, University of Michigan). *Dissertation Abstracts International, 41*(9), 3503B. (University Microfilms No. 8106101).

Cavicchio, D. J. (1970). *Adaptive search using simulated evolution.* Unpublished doctoral dissertation, University of Michigan, Ann Arbor, (University Microfilms No. 25-0199).

Davidor, Y. (1989). Analogous crossover. *Proceedings of the Third International Conference on Genetic Algorithms*, 98–103.

Davis, T. E. and Principe, J. C. (1991). A simulated annealing-like convergence theory for the simple genetic algorithm. In R. K. Belew, & L. B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 174–181).

Dawkins, R. (1986). *The Blind Watchmaker.* New York: Penguin Books.

Dawkins, R. (1976). *The Selfish Gene.* New York: Oxford University Press.

De Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems. (Doctoral dissertation, University of Michigan). *Dissertation Abstracts International 36(*10), 5140B.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms.* Chichester, UK: Wiley.

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering 186*(2–4), 311–338.

Deb, K. (1995). *Optimization for engineering design: Algorithms and examples.* Delhi: Prentice-Hall.

Deb, K. (1993). Genetic algorithms in optimal optical filter design. In E. Balagurusamy and B. Sushila (Eds.), *Proceedings of the International Conference on Computing Congress* (pp. 29–36).

Deb, K. (1989). Genetic algorithms in multimodal function optimization, *Master's Thesis*, (TCGA Report No. 89002). Tuscaloosa: University of Alabama.

Deb, K. and Agrawal, S. (1999). Understanding interactions among genetic algorithm parameters. *Foundations of Genetic Algorithms V*, 265–286.

Deb, K. and Agrawal, R. B. (1995) Simulated binary crossover for continuous search space. *Complex Systems, 9* 115–148.

Deb, K. and Goldberg, D. E. (1989). An investigation of niche and species formation in genetic function optimization, *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 42-50.

Deb, K. and Kumar, A. (1995). Real-coded genetic algorithms with simulated binary crossover: Studies on multimodal and multiobjective problems. *Complex Systems, 9*(6), 431–454.

Duffin, R. J., Peterson, E. L., and Zener, C. (1967). *Geometric Programming.* New York: Wiley.

Eldredge, N. (1989). *Macro-evolutionary Dynamics: Species, niches, and adaptive peaks.* New York: McGraw-Hill.

Eshelman, L. and Schaffer, J. D. (1993). Real-coded genetic algorithms and interval-schemata. To appear in *Foundations of Genetic Algorithms II.*

Fonseca, C. M. and Fleming P. J. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In S. Forrest (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 416–423).

Gen, M. and Cheng, R. (1997). *Genetic Algorithms and Engineering Design.* New York: Wiley.

Goldberg, D. E. (1993). A Wright-Brothers theory of genetic-algorithm flight. *Journal of SICE, 37*(8), 450–458.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* New York: Addison-Wesley.

Goldberg, D. E. (1983). Computer-aided gas pipeline operation using genetic algorithms and rule learning (Doctoral dissertation, University of Michigan). *Dissertation Abstracts International, 44*(10), 3174B. (University Microfilms No. 8402282).

Goldberg, D. E. and Deb, K. (1991). A comparison of selection schemes used in genetic algorithms, *Foundations of Genetic Algorithms*, edited by G. J. E. Rawlins, pp. 69-93.

Goldberg, D. E., Deb, K., and Clark, J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, Vol. 6, pp. 333–362.

Goldberg, D. E., Deb, K., and Thierens, D. (1991). Toward a better understanding of mixing in genetic algorithms. *Journal of SICE*, Vol. 32, No. 1.

Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. *Proceedings of the Second International Conference on Genetic Algorithms*, 41–49.

Goldberg, D. E. and Smith, R. (1987). Non-stationary function optimization using genetic algorithms with dominance and diploidy. In J. J. Grefenstette (Ed.) *Proceedings of the Second International Conference on Genetic Algorithms.* New Jersey: Lawrence Erlbaum Associates. (pp. 59–68).

Homaifar, A., Lai, S. H.-V. and Qi, X. (1994). Constrained optimization via genetic algorithms. *Simulation 62*(4), 242–254.

Harik, G., Cantú-Paz, E., Goldberg, D. E., and Miller, B. (1996). The gambler's ruin problem, genetic algorithms, and sizing of populations (IlliGAL Report No. 96004). Urbana: University of Illinois at Urbana-Champaign. Illinois Genetic Algorithms Laboratory.

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor: University of Michigan Press.

Horn, J., Nafploitis, N. and Goldberg, D. (1994). A niched Pareto genetic algorithm for multi-objective optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pp. 82–87.

Kelly, J. D. and Davis, L. (1991). Hydridizing the genetic algorithm and the K nearest neighbors. In R. Belew and L. B. Booker (Eds.) *Proceedings of the Fourth International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann. (pp. 377–383).

Kita, H., Ono, I. and Kobayashi, S. (1998). The multi-parent unimodal normal distribution crossover for real-coded genetic algorithms. Tokyo Institute of Technology, Japan.

Krishnakumar, K. (1989). Microgenetic algorithms for stationary and non-stationary function optimization. *SPIE Proceedings on Intelligent Control and Adaptive Systems, 1196*, 289–296.

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs.* Berlin: Springer-Verlag.

Michalewicz, Z. and Attia, N. (1994). Evolutionary optimization of constrained problems. In *Proceedings of the Third Annual Conference on Evolutionary Programming*, pp. 98–108.

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

Miller, G. F., Todd, P. M., and Hegde, S. U. (1989). Designing neural networks using genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann. (pp. 379–384).

Mitchell, M. (1996). *Introduction to Genetic Algorithms.* Ann Arbor: MIT Press.

Oates, M., Corne, D. and Loader, R. (1999). Variation in EA performance characteristics on the adaptive distributed database management problems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, pp. 480–487.

Powell, D. and Skolnick, M. M. (1993). Using genetic algorithms in engineering design optimization with nonlinear constraints. In S. Forrest (Ed.) *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, CA: Morgan Kaufmann (pp. 424–430).

Prügel-Bennett, A. and Rogers, A. (2001). Modelling genetic algorithm dynamics. In L. Kallel, B. Naudts, and A. Rogers (Eds.) *Theoretical Aspects of Evolutionary Computing.* Berlin, Germany: Springer, 59–85.

Reklaitis, G. V., Ravindran, A., and Ragsdell, K. M. (1983). *Engineering optimization methods and applications.* New York: John Wiley and Sons.

Rowe, J. E. (2001). The dynamical systems model of the simple genetic algorithm. In L. Kallel, B. Naudts, and A. Rogers (Eds.) *Theoretical Aspects of Evolutionary Computing.* Berlin, Germany: Springer, 31–57.

Rudolph, G. (1994). Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Network.* (pp 96–101).

Shapiro, J. (2001). Statistical mechanics theory of genetic algorithms. In L. Kallel, B. Naudts, and A. Rogers (Eds.) *Theoretical Aspects of Evolutionary Computing.* Berlin, Germany: Springer, 87–108.

Spears, W. M. and De Jong, K. A. (1991). An analysis of multi-point crossover. In G. J. E. Rawlins (Eds.), *Foundations of Genetic Algorithms* (pp. 310–315). (Also AIC Report No. AIC-90-014).

Srinivas, N. and Deb, K. (1995). Multiobjective function optimization using nondominated sorting genetic algorithms, *Evolutionary Computation, 2*(3), 221–248.

Starkweather, T., McDaniel, S., Mathias, K., Whitley, D., and Whitley, C. (1991). A comparison of genetic scheduling operators. In R. Belew and L. B. Booker (Eds.) *Proceedings of the Fourth International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann. (pp. 69–76).

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 2–9).

Thierens, D. and Goldberg, D. E. (1993). Mixing in genetic algorithms. In S. Forrest (Ed.) *Proceedings of the Fifth International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann. (pp. 38–45).

Tsutsui, S., Yamamura, M. and Higuchi, T. (1999). Multi-parent recombination with simplex crossover in real-coded genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, pp. 657–664.

Vose, M. D. (1999). *Simple Genetic Algorithm: Foundation and Theory.* Ann Arbor, MI: MIT Press.

Vose, M. D., and Liepins, G. E. (1991). Punctuated equilibria in genetic search. *Complex Systems, 5*(1). 31–44.

Whitley, D. (1992). An executable model of a simple genetic algorithm. In D. Whitley (Ed.), *Foundations of Genetic Algorithms II.* (pp. 45–62).