Genetic and evolutionary systems

Robert E. Smith

The University of Alabama
Department of Engineering Science
and Mechanics
Tuscaloosa, Alabama 35487

Abstract

Genetic algorithms (GAs) are a class of programs that emulate the search processes of natural genetic evolution. This article reviews GA technology. The review begins by recasting search as a process of sampling points from an unknown space. With this "black-box" view of search, the simple mechanics of the GA are shown to have powerful effects. A review of basic GA mechanics and theory is followed by an overview of more advanced GA techniques. Final comments address future directions for GAs and for general evolving systems research.

1 Introduction

One who works with computers is often tempted to assign life-like qualities to computational entities. Computer users speak of "daemons" that "hunt" and "munch" data, "talk" to one another, and "spawn" others like themselves. *Evolutionary computation* (EC) (De Jong & Spears, 1993) is a broad field that takes this analogy to its logical conclusion. In EC systems, populations of computational entities survive, breed, and change in a progression towards a computational goal. This article will overview how systems of evolving computational entities behave, and how that behavior can be used for practical ends.

Actually, EC is the umbrella term for three fields:

- *Evolution Strategies* (ES) (Rechenberg, 1973),
- *Evolutionary Programming* (EP) (Fogel et al., 1966), and
- *Genetic Algorithms* (GAs) (Holland, 1975).

These fields differ in many implementation details, and somewhat in philosophy. However, in recent years, the three areas have begun to influence one another. EC is becoming a more unified concept. An overview of the differences between ES, EP, and GA is given by Bäck and Schwefel (1993).

This review will focus on literature most often associated with GAs. However, all of the EC sub-fields share common characteristics: their population basis, and their use of operations that are inspired by natural genetic evolution. Most of the principles discussed here are generally applicable to EC as a whole.

GA-based programs have been applied to problems in optimization, machine learning, and evolving system modeling. Specific application areas include:

- VLSI layout

- Communications network design
- Composite materials design
- Jet turbine design
- Operations Research
- and scores of other areas.

As De Jong (1992) points out, thinking of GAs strictly as optimizers may be an oversight. GAs can have significant uses outside of optimization. However, by considering general optimization problems, one can easily see the power that evolving systems derive from simple operations and a population basis. Therefore, the following sections will introduce the GA via optimization in unknown spaces.

2 Reformulating Search

Consider being given a space of possible designs (say, designs for aircraft wings, manufacturing procedures, programs, etc.). Assume that each design can be specified by a series of features. These features can include discrete, non-numeric quantities (like color or texture), and continuous quantities like dimensions. Assume that one can determine a net utility (called *fitness*) for each design. This fitness measure can be noisy and approximate.

One wishes to find better (or, eventually, optimal) designs, given the following limitations:

- The space is far too large to examine every possible design.
- Little or nothing is known about the relationship between features and fitness.

The former limitation is simply the nature of any meaningfully large search problem. The later is more significant. It indicates that one cannot assume any particular functional form of the variations of fitness. In particular, fitness may not have smooth variations with respect to the parameters.

Given these limitations, one must formulate a method for locating highly fit designs. Clearly, one must sample the search space as a part of any reasonable approach to this black-box search problem. Given a sample of possible designs, how can one bias the selection of subsequent samples, such that increasingly fit designs are found?

One straight-forward answer to this question is to exploit apparent correlations between features and fitness values. Such correlations are implied by similarities between high-fitness designs. For example, consider these two lists of design features:

`(blue, thick, 0.1, rubber, B, 24)`
and
`(red, thick, 0.1, leather, C, 42)`

If both of the corresponding designs have relatively high fitness, then it is reasonable to assume that `thick` and `0.1` are features that should be tried in subsequent design samples. It seems reasonable that these apparently "good" features should be combined with other apparently "good" features in new designs.

To more carefully examine this approach, consider (without loss of generality) a binary encoding of possible designs. Regions of the bit string can represent finite-precision encodings of continuous-parameter features, or actual discrete features. A binary encoding of length ℓ can represent 2^ℓ possible designs. Note that the assumption of a binary encoding here is only one of convenience. One can make arguments similar to those made here for any encoding.

To formalize the notion of similarities in encoded designs, consider defining possible *similarity templates* with the alphabet $\{0, 1, *\}$, where * is a don't care, which matches either 0 or 1, e.g.:

```
* * 1 1 0 0 1
```

represents the set of strings:

```
0 0 1 1 0 0 1
0 1 1 1 0 0 1
1 0 1 1 0 0 1
1 1 1 1 0 0 1
```

In the GA literature, such similarity templates are called *schemata* (*schema* singular).

Schemata define subsets of designs that should be examined for fitness correlations. However, for 2^ℓ possible designs, there are 3^ℓ possible schemata that one might choose to examine! Clearly, examining all schemata serially and explicitly is ridiculous. One might as well examine all possible designs. However, in the limited-information search problem presented here, these similarities contain the only exploitable information available. Therefore, one needs a search procedure that examines a large number of schemata (similarities) in an efficient way. As the following section illustrates, this is what implicitly takes place in a GA.

3 Genetic Algorithm Basics

GAs search by starting with a population of structures that represent encodings (usually binary) of designs. The population can be thought of as a sample from a black-box search space. A series of GA populations (samples) are generated by the application of *genetic operators*. Typically, three operators are employed:

Selection (also known as Reproduction),

Recombination (also known as Crossover), and

Mutation.

Selection is the GA version of survival of the fittest. Structures in the current population are selected for high fitness, and placed in a *mating pool*. *Roulette-wheel* *selection* is perhaps the easiest method to introduce. In roulette wheel selection, each structure receives a copy in the mating pool with probability

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$$

where f_i is the fitness of structure i, n is the population size, and the sum is taken over all structures in the current population. These probabilities represent unequal slices of a roulette wheel. The wheel is spun repeatedly, until the mating pool is the size of the population. Selection is the phase of the GA where fitness information in the current population is emphasized. A large number of selection methods exist (Baker, 1987; Goldberg & Deb, 1990), many of which are technically preferable to roulette wheel selection. However, roulette wheel most easily illustrates selection's effects.

Structures selected for the mating pool are recombined to form new structures. A common and easily introduced recombination method is *single-point crossover*. Two mates are selected from the pool at random. A *crossover point* between 1 and $\ell - 1$ is selected uniformly at random. Bits on either side of the crossover point are swapped between parents to form a new child. For instance, consider the following two mates

```
1 1 0 1|1 0
0 1 0 0|1 1
```

If a crossover point of four is selected (as marked by the | character above), the following two children are generated:

```
1 1 0 1|1 1
0 1 0 0|1 0
```

Crossover is generally applied with a probability p_c, which is typically near one. Mates who do not cross are simply copied to their children intact. Note that under crossover, each "child" contains information from both parents. Recombination is the exploration phase of the GA. It involves a randomized operation, but it has a structured effect, which will be discussed later.

Selection and recombination are typically thought to account for most of the GA's search power. However, the ES and EP communities typically assign more importance to the role of mutation in EC systems. In most of the GA literature, mutation is thought to act primarily as an "insurance policy" to keep the population diverse during the search. The most common GA mutation technique is *point mutation*. Under point mutation, each bit of each new individual is considered separately. With a small probability p_m, the bit's value is changed.

4 GA Foundations

The basic GA procedures outlined above are simple to describe, and trivial to compute. One must wonder how these simple procedures can result in interesting computational behavior. To show how this happens, one must consider the effects of GAs on schemata, a task undertaken in the following sections.

4.1 The Fundamental Theorem of GAs

To derive the effects of the GA on similarities, one must consider some generalized characteristics of schemata. Consider the schemata

$$H_1 = * \ * \ 1 \ 0 \ * \ *$$
$$H_2 = * \ 1 \ * \ 0 \ 0 \ *$$
$$H_3 = * \ * \ * \ 1 \ * \ *$$

Schema order, $o(H)$ is the number of defined bits in a schema. For example, $o(H_1) = 2, o(H_2) = 3$, and $o(H_3) = 1$.

Schema defining length, $\delta(H)$ is the number of possible cross points between outermost defined bits. For example $\delta(H_1) = 1, \delta(H_2) = 3$, and $\delta(H_3) = 0$.

Schema average fitness, $f(H)$ is the average fitness of designs that share similarity H.

Consider roulette wheel selection effects on a single structure. The "net area" on the roulette wheel for a given structure i is the number of copies of i in the population, $m(i,t)$, times the structure's fitness f_i. Its probability of selection in one "spin" of the wheel is

$$m(i,t) * \frac{f_i}{\sum_{j=1}^{n} f_j}.$$

A complete GA mating pool is generated in n spins. In n spins, the expected number of "wins" for i is

$$n * m(i,t) * \frac{f_i}{\sum_{j=1}^{n} f_j} = m(i,t)\frac{f_i}{\bar{f}},$$

since

$$\bar{f} = \frac{\sum_{j=1}^{n} f_j}{n}.$$

For a schema H the area on the wheel is the sum of areas for all strings that share the similarity H. There are $m(H,t)$ such strings. For n spins, the expected number of "wins" for H is

$$n^* \frac{\sum_{i \in H} f_i}{\sum_{j=1}^{n} f_j} = m(H,t)\frac{f(H)}{\bar{f}},$$

since

$$\frac{\sum_{i \in H} f_i}{m(H,t)} = f(H).$$

A schema H tends to survive crossover if the crossover point falls outside its outermost defining bits. This is a conservative assumption, since it is possible that mates may share some bits of the schema, increasing its survival probability. Therefore, schema H survives crossover with (at least) probability

$$\left(1 - p_c\left(\frac{\delta(H)}{\ell - 1}\right)\right).$$

A schema H tends to survive mutation with probability

$$(1 - p_m)^{o(H)},$$

which can be approximated (through a binomial expansion and an assumption of small p_m) as

$$(1 - (p_m o(H)))\,.$$

If one combines these terms, one can derive *the fundamental theorem of GAs*, also known as *the schema theorem* (Holland, 1975):

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

where $m(H, t)$ is the expected number of structures containing schema H in the population at time t.

4.2 The Building Block Hypothesis

When one considers iteration of the GA population generation procedure, the leading (selection) term in the schema theorem indicates exponential growth for consistently above average schemata, and exponential decay of consistently below average schemata. The loss terms associated with crossover and mutation represent a filter that removes some schemata from this exponential selective process. The schemata that are emphasized (or de-emphasized) by selection are those that pass through this filter with high probability. These schemata are *short* (in terms of defining length) and *low order*. Short, low order schemata, particularly those with above average fitness, are referred to as *building blocks*.

Building blocks are the subset of the 3^{ℓ} schemata that the GA uses as a basis for search. In effect, the GA emphasizes building blocks through selection, and then juxtaposes them through recombination. This leads to the central assumption in the GA approach: *the building block hypothesis*. Simply put, it is hypothesized that combining small building blocks leads to highly fit, larger building blocks, and eventually highly fit complete designs.

4.3 Implicit Parallelism

Clearly, not all building blocks will conform to the building block hypothesis. However, in a length ℓ encoding, there are a large number of building blocks, and a large number of ways that the GA can combine building blocks towards good solutions. Therefore, it is important to consider how many building blocks the GA examines without significant disruption from crossover and mutation. These are the schemata on which the GA selective and recombinative processes operate (Goldberg, 1989b; Holland, 1975).

Consider schema that survive crossover with probability p_s or greater:

$$p_s \leq 1 - p_c \frac{\delta(H)}{\ell - 1}$$

Let $\delta(H)$ that satisfies this constraint be the schemata *survival length*, ℓ_s:

$$\frac{(1 - p_s)(\ell - 1)}{p_c} \geq \ell_s\,.$$

Goldberg (Goldberg, 1989b) graphically illustrates that a slight undercount of the number of schemata with $\delta(H) \leq \ell_s$ in a single string is

$$2^{(\ell_s-1)} * (\ell - \ell_s + 1)$$

In this argument, and undercount is desirable. This development, originally presented by Holland (1975), derives a *lower bound* on the number of schemata the GA usefully processes. Therefore, each approximate step of the development will consider an undercount of schemata.

In a population of n strings, one could assume there are

$$n * (2^{(\ell_s-1)} * (\ell - \ell_s + 1))$$

usefully processed schemata. However, this is clearly an overcount. Unfortunately, it is difficult to say anything about the representation of schemata in terms of population size, without knowing population size as a function of the desired survival length, ℓ_s.

The probability of a random string being a member of some schema H is

$$p = \left(\frac{1}{2}\right)^{o(H)}$$

The probability of m random strings in a population of size n being members of H is

$$p(m,n) = \left(\begin{array}{c} m \\ n \end{array}\right) * p^m * (1-p)^{(n-m)}.$$

The expected number of strings from a random population of size n in a schema H is the expected value of a binomial distribution:

$$E[m] = np = n * \frac{1}{2}^{o(H)}$$

To insure an undercount, only consider schemata with $o(H) \geq \ell_s/2$. These schemata are less likely to be duplicated in the population. Setting this expected number to one:

$$1 = n * ((1/2)^{(\ell_s/2)})$$
$$n = 2^{(\ell_s/2)}$$

Assuming this population size counts only the schemata of order $\ell_s/2$ or greater. Half the schemata in the previous count for a single string are of order greater than $\ell_s/2$. Thus, the count must divide the schemata count by one-half:

$$(2^{(\ell_s-1)}) * (\ell - \ell_s + 1)$$

Multiplying by n:

$$n_s = n * (2^{(\ell_s-1)}) * (\ell - \ell_s + 1)$$

and substituting $n = 2^{(\ell_s/2)}$ yields:

$$n_s = \frac{((\ell - \ell_s + 1) * (n^3))}{4} = C * (n^3)$$

Thus, the number of schemata that survive crossover to be evaluated, selected, and recombined is at least $O(n^3)$ when n structures are processed.

Of course, the $O(n^3)$ estimate is based on a population size assumption. As will be shown in a later section, the $O(n^3)$ argument does not imply that one should use as large a population as possible. For population sizes other than that selected by Holland to derive a lower-bound schemata count, the $O(n^3)$ estimate is not valid. However, the $O(n^3)$ argument does show that (for moderate population sizes) the GA examines far more schemata than the population size. This examination goes on implicitly, and in parallel, and accounts for the computational leverage of the GA approach. The effect of processing this large amount of information is called *implicit parallelism*. Implicit parallelism shows that **although all building blocks do not conform to the building block hypothesis, the GA looks at a large number of them. The GA may find one of many paths to a highly fit solution.** Empirical evidence shows that this approach is effective in a broad range of black-box search problems.

4.4 The K-Armed Bandit

Previous arguments have established that the GA processes schemata (in particular, building blocks), and that the GA looks at a large number of schemata implicitly and in parallel. In effect, the GA is making a large number of decisions between schemata. Each schema belongs to a *partition* of schemata that have the same defined bits. One can represent partitions with the alphabet {f, *}. For instance, the partition

```
* * f * f *
```

contains schemata

```
* * 0 * 0 *
* * 0 * 1 *
* * 1 * 0 *
* * 1 * 1 *
```

In selecting between strings, the GA is implicitly selecting between schemata in short, low order partitions. The GA makes this decision (implicitly) on the average fitness of each schema, as indicated by a finite sample of strings that share that schema. The GA makes this decision in a gradual fashion, that exponentially builds towards assigning all individuals to a given schema within each partition. It is fair to ask whether this exponential relationship between exploration and exploitation is appropriate. Holland (Holland, 1975) examines this relationship through a problem in decision theory: the *k-armed bandit*. In this problem, one is presented with k "arms" of a slot machine. Each returns a payoff with a mean μ_i and variance σ_i. One is to gamble at this machine N times. It is assumed that one will try each arm an equal number of times, n, (for exploration). Then, one will play the apparent maximum payoff arm the remaining number of times, $N - kn$, (for exploitation). One wishes to find the optimal value for n that minimizes losses. The k-armed bandit argument (not presented here for the sake of brevity) shows that the optimal number of exploitative trials expands exponentially with the number of exploratory trials. This argument provides some justification for the exponential allocation of trials in GAs.

The building block hypothesis is the central assumption of the GA. This assumption is supported by the fundamental theorem, implicit parallelism, and the k-armed bandit argument. However, one must more carefully consider cases where the building block hypothesis is incorrect.

4.5 Deception

Clearly, it is possible for the building block hypothesis to be incorrect for some partitions. Consider the following case:

```
f(**1**...**) > f(**0**...**)
f(*****...1*) > f(*****...0*)
```

where these are the average fitness values of schemata in partition `**f**...**` and `*****...f*`, respectively. If the building block hypothesis works in this case, then the partition `**f**...f*` should behave such that

```
f(**1**...1*) > f(**0**...0*)
              and
f(**1**...1*) > f(**0**...1*)
f(**1**...1*) > f(**1**...0*)
```

However, the situation could exist that if

```
f(**1**...1*) < f(**0**...0*)
              and
f(**0**...0*) > f(**0**...1*)
f(**0**...0*) > f(**1**...0*)
```

In this case, one would say that the lower order (partitions `**f**...**` and `*****...f*`) are *deceptive* with respect to the higher order partition (`**f**...f*`).

Deception means that the lower order partitions point the GA in the wrong direction with respect to higher order partitions. However, this may not mean that the GA will ultimately be drawn away from the best schemata in the higher order partitions. First of all, if the higher order partition is short and low order enough to allow its schemata to serve as building blocks, deception in the lower order schemata may be irrelevant. Moreover, the GA looks at many building blocks (through implicit parallelism). Deception in one partition may be circumvented by effective search in another, overlapping partition.

It is possible to develop problems with pathological deception. Fully deceptive problems have been developed where every partition points away from the ultimately optimal string (Goldberg, 1989a). Deception is of great theoretical interest in the GA community. However, it is important not to overestimate full deception's practical effects on GA efficacy. In practice, fully deceptive problems are unlikely to occur. Partial deception is more likely, and may present difficulty in some GA problems. However, one must note that even in partially deceptive problems, the GA can find highly fit solutions. Partial deception may prevent the GA from finding the ultimate optima in some situations. However, in many problems finding highly fit solutions is sufficient. In such problems, the GA is an approach that requires little problem specific knowledge, and yields high-quality results in a large number of problems.

5 Advanced Theory

The previous analysis justifies the GA approach, but leaves several implementation questions unanswered. The following sections present recent theoretical developments that begin to answer some of these questions.

5.1 Population Size

How large should a GA population be for a given problem? By misinterpreting the $O(n^3)$ argument, one might conclude that the larger n is, the better the GA will perform. This is clearly not the case, since there are only a finite number of schemata. One clearly cannot process $O(n^3)$ schemata if n^3 is much greater than 3^ℓ. Goldberg (1989c) shows how to set population size for an optimal balance of computational effort to computational leverage. This development is outlined below.

One can derive an exact count of the number of expected number of schemata in a population of size n given strings of length ℓ:

$$S(n,l) = \sum_{i=0}^{\ell} \binom{\ell}{i} 2^i \left\{ 1 - \left[1 - \left(\frac{1}{2}\right)^i \right]^n \right\}$$

Combining this with the argument used earlier can yield a slight undercount of the number of schemata of length ℓ_s:

$$n_s(n, \ell, \ell_s) = (\ell - \ell_s + 1)\, S(n, \ell_s - 1)$$

For an appropriate survival length, this gives a count of the number of building blocks for a given population size. Note that the number of building blocks monotonically expands from $(\ell - \ell_s + 1)\, 2^{\ell_s}$ (for a population of one) to $(\ell - \ell_s + 1)\, 3^{\ell_s}$ (for an infinite population).

Goldberg assumes that a useful measure of the GA's computational leverage is dS/dt, the average real-time rate of schema processing. Assume that the GA starts with a random population, where the number of building blocks is S_0. Further assume that the population ultimately converges to contains only one unique string, and thus, 2^ℓ schemata, Therefore,

$$\frac{dS}{dt} = \frac{(S_0 - 2^\ell)}{Dt},$$

where Dt is the time to convergence. Assume

$$Dt = n_c t_c$$

where n_c is the generations to convergence, and t_c is the real computational time per generation. Goldberg shows that convergence time (with fitness proportionate reproduction) is either $n_c = O(1)$ (constant with respect to n), or $n_c = O(\ln n)$, depending on what degree of convergence is assumed.

The time t_c varies with the degree of parallelization, since parallel computers can evaluate several fitness values simultaneously. Therefore,

$$t_c = n^{(1-\beta)}$$

The value $\beta = 1$ represents a perfectly parallel computer, where all fitness values are evaluated at once. The value $\beta = 0$ represents a perfectly serial computer. Note that Dt increases monotonically with n.

Goldberg maximizes DS/Dt with respect to population size. This gives a series of curves that indicate the best population size for maximum average schemata processing rate. These curves indicate that there is a finite population size that maximizes this GA performance measure.

Surprisingly, Goldberg's development for serial computers and the $O(\ln n)$ convergence assumption indicated that one should use the smallest population possible, and repeatedly restart the GA after convergence. This inspired the *micro-GA* (Krishnakumar, 1989), which uses very small populations, and repeated, partially random restarts. Although results of the micro-GA are promising, it is important to note that the optimal population size for schema processing rate *may not* be the optimal size for ultimate GA effectiveness. *Sampling error* may overwhelm the GAs ability to select correctly in small populations.

A recent study (Goldberg et al., 1991) examines population sizing in terms of ultimate GA performance by considering sampling error. This examination focuses on the fitness variance in each schema partition. For each schema, the fitness varies depending on what bits are in the $*$ positions. Therefore, each schema has mean fitness $f(H)$, and a fitness variance $V(H)$, taken over *all* strings that contain H. For a given population, a finite sample of strings match schema H. This sample has an observed average fitness $f_o(H)$. The GA selects based on $f_o(H)$. This observed fitness $f_o(H)$ is normally distributed with mean $f(H)$ and variance $V(H)/m(H)$. Variance of observed schemata average fitness values is a function of (H),which is in turn a function of population size. This variance can cause selection error.

Assume $f(H_1) > f(H_2)$, where H_1 and H_2 are in the same partition. Consider the probability of a selection error, $p(f_o(H_1) < f_o(H_2))$. The distribution of $f_o(H_1) - f_o(H_2)$ is normal, with mean $f(H_1) - f(H_2)$ and variance $(V(H_1)/m(H_1)) + (V(H_2)/m(H_2))$. Given this distribution, one can size the population for a desired probability of selection error. Goldberg (1991) derives population sizing relationships for several analytically derived example problems.

Although the sampling-based population sizing formulas Goldberg develops are informative, they are difficult to apply in practice. To size the population, one must know schemata fitness and variance values. If one knew these, one would not have to use a GA (or any search technique) at all. However, in a recent study (Smith, 1993a; Smith, 1993b), a modified GA is suggested that adaptively resizes the population. This resizing is based on Goldberg's calculations, and uses competitions of mates to estimate not only schema average fitness (as in the usual GA), but schemata fitness variances as well. Ongoing studies show that this technique is effective in solving problems that are known to present significant sampling error problems to the GA (Smith & Ideh, 1994).

5.2 Representation

The binary representation of designs is most frequently associated with the GA, but it is not necessarily one of the GAs defining features. Applications in GAs, EP, and ES have all used a variety of representations. However, there are some technical reasons to believe that, in black-box search problems, binary is preferable (Goldberg, 1990b)

Consider a search space with C points and an encoding alphabet of cardinality k. Note that a binary representation with any restriction on crossover points is effectively a representation with $k > 2$. For any cardinality k, the encoding length ℓ must be such that

$$k^{\ell} = C.$$

The number of schemata in this encoding is

$$S = (k+1)^{\ell}$$

where schemata are from the original alphabet with the $*$ character added. From the previous calculation

$$\ell = \frac{\log(C)}{\log(k)}.$$

Therefore

$$S = (k+1)^{\frac{\log(C)}{\log(k)}}.$$

For a given space size C, S decreases monotonically with cardinality. This indicates that the maximum number of schemata occurs for $k = 2$. More importantly, this also indicates that the maximum number of building blocks occurs in a binary encoding.

Some new arguments on cardinality have been presented in recent years. Antonisse (1989) indicates that the count should be refined. Consider a $k = 3$ alphabet of $\{0,1,2,3\}$. One could define other schemata by including wild card characters that define limited subsets of the alphabet. For instance:

$$\begin{aligned} * &= \{0,1,2\} \\ *_{02} &= \{0,2\} \\ *_{01} &= \{0,1\} \\ &\text{etc.} \end{aligned}$$

The schema theorem applies for these new schemata, and to all other subsets of the search space. If one counts the new schemata, binary has far fewer.

However, one must take care in counting these new schemata. In a binary encoding, all higher order schemata can be built from lower order schemata through recombination. Thus, these low order schemata can act as building blocks. However, some of the subsets indicated by the new schemata cannot be acted on in the same fashion. For instance, consider two bits that encode the numbers zero through three. The schemata $*0$ and $1*$, when represented by 00 and 11 can recombine to form 10. Now consider a cardinality four encoding. Analogous subsets $*_{02}$ and $*_{23}$, when represented by the analogous strings 0 and 3, cannot recombine to form the analogous string 2. In effect, these schemata's function as building blocks is not identical to that of traditional schemata. Since building blocks are the functional units of the GA, counting all of the extra schemata may give inaccurate results.

One can make another argument for higher-cardinality alphabets. Although binary may have the largest number of functional building blocks, some of these building blocks may not have any particular meaning in the problem at hand. For instance, if one decodes a string of bits as a decimal integer, the schema $**\ldots 0$ represents the concept "even". Even valued parameter settings may have no particular meaning to the problem at hand. Moreover, higher cardinality alphabets have been used with positive results in many cases (Davis, 1991). In cases where one knows about the internal structures of the problem, a high cardinality alphabet may be advantageous. However, if the problem is a black box, binary still seems to have a theoretical advantage.

5.3 Selection

As was pointed out earlier, roulette wheel selection, while being easy to introduce, has some disadvantages. First, its results have a high variance (Baker, 1987). Second, as fitness values converge to a close range, pressure towards the observed best schemata is small under roulette wheel selection. This problem is called "poor scaling". Literal rescaling of fitness values is sometimes used to overcome this problem (Goldberg, 1989b).

Alternate selection schemes include other fitness proportionate methods:

- stochastic universal selection (SUS) (Baker, 1987)
- stochastic remainder selection (SRS) (Booker, 1982; Brindle, 1981)

These variants are suggested to overcome the noise of roulette wheel selection.

To overcome poor scaling, alternate selection methods include those based on *ranking* (Baker, 1985). In ranking methods, one sorts the population on fitness, and assigns each a number, based on rank. Then one performs proportionate selection on the assigned values.

Another popular method is tournament selection. In this method, one breaks the population into random tournaments of size k. One then assigns the best individual in each tournament a copy in the mating pool. This procedure is repeated k times. Tournament selection has the advantages that it is easily parallelized, and that it automatically overcomes the scaling problem. Under tournament selection, the best individual always receives k copies in the mating pool. Tournament selection with $k = 2$ is the most commonly employed.

Another popular method is *genitor*, (Whitley, 1989a) or *block selection*. Two parents are picked by ranking. One child is produced. The worst individual in the population is replaced (deterministically) by this child. One must perform n such cycles to equal a regular GA generation.

With the proliferation of selection schemes, one must wonder which is best. This is not entirely clear from the literature. An analytical comparison is given by Goldberg and Deb (1990). The strongest conclusion one can draw about selection is that it must be used cautiously. Given that the GA is involved in a gradual process of selecting based on noisy, observed fitness values, one must select gradually.

An interesting new development in GA selection is its combination with simulated annealing (SA). SA is a search procedure based on metal annealing (Kirkpatrick et al., 1983). SA's advantage is that it is a tuned selection scheme. A parameter called temperature is slowly lowered in the SA process. At a given temperature, the search points visited by the SA procedure appear according to the Boltzmann probability distribution. As temperature is lowered, the shape of this distribution shifts towards the global optima. If temperature is tuned slowly enough, one can be assured that SA will find the global optima. Several papers have tried to use a population based form of SA as a selection scheme. Early efforts are reviewed by Goldberg (1990a). A promising new SA-selection method is suggested by Mahfoud and Goldberg (1992). This method can be shown to possess the global convergence properties of SA.

5.4 Recombination

Like roulette wheel selection, single point crossover is often used in GA introductions because of its analytical ease. Many other crossover methods exist. These include multipoint crossover schemes (Spears & De Jong, 1991). In GAs with higher cardinality alphabets, a wide variety of recombination methods exist. A popular recombination method is uniform crossover (Syswerda, 1989). Under uniform crossover, each bit for a child is randomly selected from one parent with probability p_s, and from the other parent with probability $1 - p_s$. Typically, $p_s = 1/2$.

Spears and De Jong (Spears & De Jong, 1991) offer an analytical examination of various recombination schemes. The conclusions of this examination can be summarized as follows. Larger numbers of crossover points yield larger schema disruption rates. Even numbers of crossover points have some small advantage, in that they treat the chromosome as a *ring*. That is, schemata with defined bits near the ends of a chromosome have some enhanced survival probability, as if the chromosome had its ends linked.

The disruption rate for uniform crossover depends on order, rather than defining length. This disruption rate is generally higher than that of the crossover schemes discussed above. The disruption rate of uniform crossover can be lowered by lowering p_s. However, as Goldberg, Deb, and Thierens point out (Goldberg et al., 1993), one must also examine recombination methods for their tendency to mix building blocks, as well as their disruption rates. If a recombination method (such as uniform crossover with low p_s) has a low mixing probability, it may give inadequate solutions. The balance of mixing and selection is an active topic of GA research.

5.5 Mutation

As was mentioned earlier, mutation is generally considered to be a necessary, but secondary operator in the GA search process. In essence, mutation is a random walk in the search space. Too much mutation is thought to significantly disrupt the selection process (Smith & Goldberg, 1992). A recent study has shown how some problems can be particularly difficult for mutation-only EC systems (Horn et al., 1992). However, another study points out that the distinction between crossover and mutation may be unnecessary (Spears, 1993). The importance of mutation in GAs and other EC systems is still a subject of much debate.

6 Advanced GA techniques

The previous sections discuss the basic techniques and analysis of GAs. Advanced techniques are continually being developed to extend the GAs applicability. An overview of some of these techniques is offered in the following sections.

6.1 Sharing

The GAs discussed thus far ultimately converge to a single point in the search space. Selection will drive towards a homogeneous population, composed almost entirely of duplicates. This can be shown by considering the actions of fitness proportionate selection (Smith et al., 1993).

Let P_i^t be the proportion of individual i in the population at time t ($\sum_{i=1}^{2^\ell} P_i^t = 1$), and let f_i be the expected fitness of individual i. Under fitness-proportionate selection, the value of P_i at time $t+1$ is given by

$$P_i^{t+1} = \frac{P_i^t f_i}{\sum_{j=1}^{2^\ell} P_j^t f_j} \ , \tag{1}$$

where the denominator is the mean fitness of individuals in the population at time t.

This equation's only steady state is a population of equally fit individuals. Assuming (for analytical purposes) that the initial population contains all possible individuals in equal proportions (a *flat population*), the only steady state contains only individuals with the highest-possible fitness value. Any such distribution is a valid steady state. Therefore, a diverse group of structures with identical fitness could theoretically make up a steady-state GA population. However, in practice *genetic drift* tends to move towards a population of replicas of only one structure (Goldberg & Segrest, 1987).

One might want to use the GA's population basis to simultaneously search for several fitness peaks in the search space. There are at least two situations where one might want to locate multiple peaks

- multi-modal optimization problems
- machine learning problems, which require a diverse population to cooperate and perform a computational task (Holland et al., 1986).

To counter the effects of genetic drift, and allow for multi-modal GA search, sharing suggests the following natural analogy. Living creatures evolve to fill environmental niches, where resources are available. When the population in a niche is too high, resources aren't plentiful enough, and evolution migrates away from the niche. This eventually leads to a balance of creatures in a variety of niches.

Through this analogy, Goldberg and Richardson (1987) introduce GA *sharing*. Sharing degrades the fitness of each population member based on its "distance" from all other population members. The distance between population members i and j, d_{ij}, can be *phenotypic* (in the decoded parameter space), or *genotypic* (in the encoding space).

Sharing recalculates an effective fitness, f_i' as its basis for selection, as follows:

$$f_i' = \frac{f_i}{\sum_{j=1}^{N} Sh(d_{ij})}\ , \tag{2}$$

where d_{ij} is the distance between i and j under a given metric, $Sh(d_{ij})$ is the *sharing function*, given by

$$Sh(d_{ij}) = \begin{cases} 1 & \text{if } d_{ij} = 0 \\ 1 - \left(\frac{d_{ij}}{\sigma_s}\right)^{\alpha} & \text{if } d_{ij} < \sigma_s; \\ 0 & \text{otherwise} \end{cases}\ , \tag{3}$$

N is the number of individuals in the population, and σ_s and α are parameters. Typically, $\alpha = 1$.

Under sharing, if any individual (and its neighbors) begin to dominate the population, its fitness degrades. Thus, sharing balances to a diverse population as a steady state.

Deb (1989; 1989) shows how sharing can be effective in multi-modal optimization problems. However, setting the parameter σ_s is critical. This parameter represents the niche size (distance between peaks). If σ_s is too large, some peaks may be overlooked. In the limiting case where σ_s equals the radius of the search space, there is no net sharing effect. If σ_s is too small, peaks become "smeared" (weak multi-modal convergence). In the limiting case $\sigma_s = 0$, there is no effective search for peaks in the space. Individuals are simply distributed based on their relative fitness. Deb shows how σ_s can be adequately set for a known number of peaks that are evenly distributed throughout the search space.

Because of the critical nature of σ_s, unequally distributed peaks could cause difficulties. Another problem with sharing is that it requires an expensive ($O(n^2)$), global calculation (that of d_{ij} for all i and j). However, recent studies have shown how sharing effects can emerge in certain GA-based immune system simulations (Smith et al., 1993). In machine learning systems based on GAs, like the learning classifier system (Holland et al., 1986), emergent sharing effects may preclude the complications of explicit sharing.

6.2 Diploidy and Dominance

Baker and Farrell (1992) point out an interesting distinction between optimization, adaptation, and learning. In their framework, optimization is tuning parameters to a given situation, adaptation is perpetually tuning parameters to changing situations, and learning is memorizing information as adaptation occurs. GAs considered thus far only perform optimization. Sharing allows for some long-term adaptation in the GA, but only of a particular form.

Consider GAs applied to nonstationary search problems where the distribution of fitness in the parameter space changes as the GA operates. Recall that the GA searches by processing $O(n^3)$ building blocks. However, this estimate is based on a random population. Clearly, as convergence occurs, the number of building blocks processed drops. Although convergence is desirable, if the function changes after convergence, there is insufficient diversity to continue to a GA search. One could simply raise the mutation rate, but this adds noise to the selection-recombination process. Cobb and Grefenstette (1993) have introduced a scheme where increased mutation rates are triggered to allow the GA to re-adapt as fitness conditions change.

Because of limited diversity after convergence, simple GAs have fared poorly on time varying functions (Pettit & Swigger, 1983). The GAs discussed thus far have used *haploid* representations. However, many higher animals use *diploid* representations that incorporate redundant information. In diploid strings, the redundant information is decoded to a unique, effective string, through a dominance relationship. The extra decoding step resolves conflicts between the redundant diploid strings.

Biological theory suggests that the extra complexity of diploidy and dominance help species to cope with time varying environments. Similar effects can be obtained in diploid GAs. Studies by Smith and Goldberg (Smith, 1988; Smith & Goldberg, 1992) consider diploid GAs applied to time varying problems. The diploid GAs show a substantial advantage over haploid GAs for such problems. Analysis accounts for this advantage in two ways. One, the diploid GA preserves diversity through the abeyance of recessive alleles. This means the diploid GA requires lower mutation rates, as was indicated by Holland. Moreover, the diploid GA can tolerate mutation with less disruption to the selective process. Second, the diploid GA tends to preserve diversity selectively. Bit positions that should vary to obtain time varying optima maintain alternate values longer. This is a *probabilistic memory* of periodically varying environmental conditions.

6.3 Mutli-Objective Techniques

The search problems discussed in this article thus far map a set of parameters to a point on the real line, that is, the fitness for those parameters. Optimization seeks an extrema on this line. In other problems, one might want to consider two or more objectives. For instance, in automobile designs, one might want to consider both fuel economy and accident risk.

One approach would be to assign a weight to each objective, and optimize the weighted combination. However, some quantities (like risk, in deaths per year) may be difficult to weigh in terms of other quantities (like economy, in miles per gallon). Therefore, one must consider the concept of *Pareto optimality*.

In single objective optimization, one searches for an extreme point. In n-objective optimization, one searches for an $n-1$ dimensional surface, called the *Pareto optimal front*. This front is the set of points for which no other point is *dominated* by another. That is, no point on the front is worse than another in *all* n objectives. The front can be concave or convex in several directions, depending on how the objectives are to be extremized.

In an early effort to apply GAs to multi-objective search, Schaffer (1985) introduces *VEGA, the vector evaluated GA*. In VEGA, one randomly divides the population into n subpopulations, and selects for a different objective in each subpopulation. Mating and crossover are performed across the general population. Schaffer shows that this technique does distribute points in the population on the Pareto optimal front. However, Schaffer also realized that VEGA could not promote points on the middle on the front, because these points are not outstanding in either criteria.

In a more recent study, Horn and Nafpliotis (1993) suggest a sharing based scheme for multi-objective optimization: *the niched Pareto GA*. This technique consists of:

- Pareto domination tournaments for selection
- sharing based on "equivalence classes"

In Pareto domination tournaments, one selects two individuals at random, and then a sample of k other individuals. If one of the first two individuals is dominated by anyone in the other k, and the other isn't, then the latter is selected. Otherwise, equivalence sharing is performed, to avoid drift towards one individual on the front. If the individuals tie in the domination tournament, one counts the individuals within σ_s in the population. One then selects the individual for whom this niche count is lowest.

Results for this technique applied to Schaffer's test function show that the niched Pareto GA has more uniform spread along the front. However, with the niched Pareto GA, some points are outside the non-dominated region. Horn and Nafpliotis also show results on a real-world multi-objective problem, with positive results. The technique seems to be a promising for locating points on a Pareto optimal front.

6.4 Order-Based GAs

There are many search problems that are not easily represented in the combinatorial forms discussed thus far. Consider the *traveling salesman problem* (TSP) (Goldberg & Lingle, 1985; Grefenstette et al., 1985). In this problem, one is given a map with N cities. One must devise a minimum length route such that each city is visited once and only once. This problem is not easily represented as a combination of bits. Instead, the problem is easily represented as a permutation of elements (cities).

In the TSP, one can simply represent a route as a permutation of city numbers. Many other problems are best represented in this fashion. To use a GA on this representation, one must develop recombination operators that preserve valid permutations. Many such operators have been created (Fox & McMahon, 1991; Goldberg & Lingle, 1985; Holland et al., 1985; Whitley, 1989b). Results with these order-based GAs are promising. However, only limited analysis of these techniques exist. They remain an open area of GA research.

6.5 Adaptive Recoding

As was emphasized earlier, the key to the GA's success is its ability to propagate and recombine building blocks. In the basic GA, only *tightly linked* bits (those with few crossover points between them) can act as building blocks. One must ask, what if the necessary bits aren't linked? For example, a problem could have poorly linked, critical schemata that can only be built up through smaller, deceptive building blocks. With poor linkage, these schemata will not be correctly constructed from component building blocks.

To obtain effective linkage, one can consider adaptively reordering the encoding. To consider adaptive recoding, assume all bits are tagged, such that one knows which bits correspond to which elements of the code. Once tagged, one can apply a simple reordering operator called *inversion* (Holland, 1975). In inversion, one picks two points along the string at random, and reverses the order of bits between these points. All orders can be reached by a series of inversions. Inversion is a unary operator that causes a random walk in the space of orderings. Inversion has been used as a mutation operator in order-based GAs (Fox & McMahon, 1991).

Goldberg and Bridges (1988) consider an idealized, unary reordering operator (like inversion) in an analytical setting. Their study shows that although the idealized operator can work, its effects are too slow to be reasonably effective.

Goldberg and Lingle (1985) originally suggested one of the first order-based GA recombination operators, *partially mapped crossover* (PMX), as an adaptive recoding crossover operator for combinatorial GAs. There has been little research into this use of order-based recombination operators.

The *messy* GA (mGA) (Deb, 1991; Goldberg et al., 1990) is yet another adaptive coding scheme. The mGA works by starting off with substrings (incomplete solutions), and cutting and splicing (as opposed to crossing over). In effect, this extends the GA paradigm to a long term evolution: from primordial to advanced organisms.

The mGA raises the question of how one evaluates under and overspecified GA strings. In the mGA, underspecified strings are evaluated by filling in with bits from a best-known solution. Overspecified strings are simply evaluated from right to left. In the mGA process, one initializes the population with all the building blocks of order k. The k value is selected as an estimate of the maximum order of deception. Then one selects and splices during the *primordial* phase. During the primordial phase, the population size is continually lowered. Once a small population size is reached, cut and splice operators are applied in a *juxtapositional phase*. Deb (1991) has shown that the mGA can solve problems with an arbitrary level of deception k with a computational complexity of $O(\ell^k)$. However, this assumes that one knows the level of deception k. If not, one can iterate the GA process for increasingly higher orders k. Deb (1991) shows that this technique is effective. However, it does increase the computational requirements of the mGA.

Another adaptive recoding scheme, *dynamic parameter recoding* (DPE), is suggested by Schraudolph and Belew (1991). This scheme does not concentrate on overcoming poor linkage, but instead attempts to effectively apply the discrete optimization ability of the GA to continuous search spaces. In DPE, one discretizes each continuous parameter in a search range a and b with an ℓ bit string. Each GA string is decoded to a value i. Then it is randomly mapped the value corresponding to i and the next highest discretized parameter value:

$$T(i) = a + (b - a)\frac{i + X}{2^\ell}$$

where X is a uniform random variable between 0 and 1, and T is the parameter value used for evaluating fitness. DPE assigns each parameter two bits $\ell = 2$. Then, the GA is run to convergence. After convergence, the search interval is narrowed to an interval around the new GA solution. Two bits are assigned to the new interval, and the GA is re-initialized.

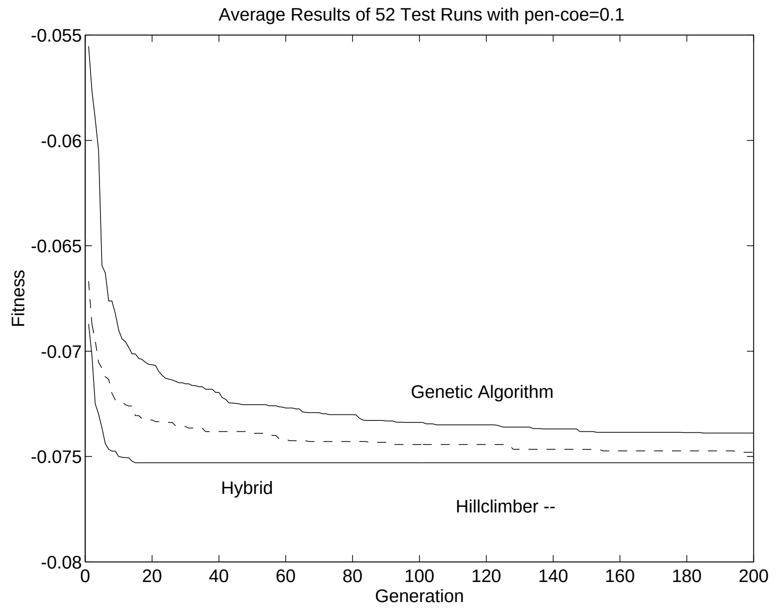

Figure 1: Comparative results of a GA hybridized with a gradient search scheme

The DPE scheme has yielded promising results. However, it is important to realize that any discretized scheme could cause the GA to be misled. An initial convergence error in DPE could cause the GA to ultimately overlook the correct final solution.

6.6 Hybridization

When one considers the GA in optimization, one is tempted to compare it directly to more traditional, gradient based optimization schemes. However, the GA operates on different sorts of information than gradient schemes. The central assumption of a gradient scheme is some smooth variation of the fitness surface with respect to parameters. The central assumption of a GA is the building block hypothesis. These assumptions are not incompatible. Recent studies (Belew et al., 1990; Smith & Huettner, 1994) show that GAs and gradient schemes can be effectively combined. The fitness of each GA individual can be evaluated by using the indicated parameters as a starting point for gradient search. In effect, the GA searches for good starting points in the overall search space. Figure 1 shows the results of such a hybrid GA averaged over fifty-two randomly generated, multi-modal minimization problems. The results show how a GA can supplement the normally local search of gradient search schemes. The results also suggest the GA as a method of improving parallelization of gradient search.

7 Future Directions

This overview has primarily focused on GA practice and theory. As with most introductions in this area, the discussion has also centered on optimization applications. However, one should note that the most interesting application of GAs and other EC systems may be outside of traditional optimization. The population basis of the GA can have particular advantages in machine learning applications. In the learning classifier system (LCSs) (Holland et al., 1986), a GA-based production system, the GA evolves a diverse population of interacting rules that jointly perform a computational task. Analogous systems can be used in neural networks (Smith & Cribbs, 1994). GAs can also be used to evolve a constantly changing set of structures that interact to explore a search space (Smith & Gray, 1994). In these applications, the fitness of each GA individual can depend on the population composition itself. Such *co-adaptive* GAs are just beginning to be formally analyzed and understood. However, such systems are yielding useful results (Dike & Smith, 1993). The development of general purpose adaptive systems, where diverse computational entities interact and breed towards a computational goal, remains a promising aim for GA and EC research.

References

Antonisse, J. (1989). A new interpretation of schema notation that overturns the binary encoding constraint. *Proceedings of the Third International Conference on Genetic Algorithms*, 86–91.

Back, T., & Schwefel, H. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation, 1*(1), 1–23.

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, 101–111.

Baker, J. E. (1987). Reducing bias and ineffiency in the selection algorithm. *Proceedings of the Second International Conference on Genetic Algorithms*, 14–21.

Baker, W. L., & Farrell, J. A. (1992). An introduction to connectionist learning control systems. In *Handbook of intelligent control: Neural, fuzzy, and adaptive approaches* (35–64). New York: Van Nostrand Reinhold.

Belew, R. K., McInerney, J., & Schraudolph, N. N. (1990). *Evolving networks: Using the genetic algorithm with connectionist learning* (CSE Tech. Report No. CS90-174). La Jolla, CA: University of California, San Diego.

Booker, L. B. (1982). Intelligent behavior as an adaptation to the task environment (Doctoral dissertation, Technical Report No. 243. Ann Arbor: University of Michigan, Logic of Computers Group). *Dissertations Abstracts International, 43(2)*, 469B. (University Microfilms No. 8214966)

Brindle, A. (1981). *Genetic algorithms for function optimization.* Unpublished doctoral dissertation, University of Alberta, Edmonton, Canada.

Cobb, H. G., & Grefenstette, J. J. (1993). Genetic algorithms for tracking changing environments. In S. Forrest (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*, 523–530, San Mateo, CA. Morgan Kaufmann.

Davis, L. (1991). *Handbook of genetic algorithms.* New York: Van Nostrand Reinhold.

De Jong, K. A. (1992). Genetic algorithms are NOT function optimizers. In L. D. Whitley (Ed.), *Foundations of genetic algorithms II* (5–18). San Mateo, CA: Morgan Kaufmann.

De Jong, K. A., & Spears, W. M. (1993). On the state of evolutionary computation. In S. Forrest (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*, 618–623, San Mateo, CA. Morgan Kaufmann.

Deb, K. (1989). *Genetic algorithms in multimodal function optimization* (TCGA Report No. 89002). Tuscaloosa: The University of Alabama, The Clearinghouse for Genetic Algorithms.

Deb, K. (1991). *Binary and floating-point optimization using messy genetic algorithms* (TCGA Report No. 91004). Tuscaloosa: University of Alabama. (Ph.D dissertation).

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

Dike, B. A., & Smith, R. E. (1993). Application of genetic algorithms to air combat maneuvering. In W. Karplus, & M. L. Padgett (Eds.), *Proceedings of the Fifth Workshop on Neural Networks: Academic/Industrial/NASA/Defense*, Vol. 2204, 84–94, San Diego. SPIE.

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). *Artificial intelligence through simulated evolution.* New York: John Wiley.

Fox, B. R., & McMahon, M. B. (1991). Genetic operators for sequencing problems. In G. Rawlins (Ed.), *Foundations of genetic algorithms* (284–300). San Mateo, CA: Morgan-Kaufmann.

Goldberg, D. E. (1989a). *Genetic algorithms and Walsh functions: Part II, deception and its analysis* (TCGA Report No. 89001). Tuscaloosa: The University of Alabama, The Clearinghouse for Genetic Algorithms.

Goldberg, D. E. (1989b). *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison–Wesley.

Goldberg, D. E. (1989c). Sizing populations for serial and parallel genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*, 70–79.

Goldberg, D. E. (1990a). *A note on Boltzmann tournament selection for genetic algorithms and population–oriented simulated annealing,* (TCGA Report No. 90003). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.

Goldberg, D. E. (1990b). *Real-coded genetic algorithms, virtual alphabets, and blocking* (IlliGAL Technical Report No. 90001). Urbana, Illinois: University of Illinois at Urbana-Champaign.

Goldberg, D. E., & Bridges, C. L. (1988). *An analysis of a reordering operator on a GA–hard problem* (TCGA Report No. 88005). Tuscaloosa: The University of Alabama, The Clearinghouse for Genetic Algorithms.

Goldberg, D. E., & Deb, K. (1990). *A comparative analysis of selection schemes used in genetic algorithms* (TCGA Report No. 90007). Tuscaloosa: The University of Alabama, The Clearinghouse for Genetic Algorithms.

Goldberg, D. E., Deb, K., & Clark, J. H. (1991). *Genetic algorithms, noise, and the sizing of populations* (IlliGAL Technical Report No. 91010). Urbana, Illinois: University of Illinois at Urbana-Champaign.

Goldberg, D. E., Deb, K., & Korb, B. (1990). *An investigation of messy genetic algorithms* (TCGA Report No. 90005). Tuscaloosa: The University of Alabama, The Clearinghouse for Genetic Algorithms.

Goldberg, D. E., Deb, K., & Thierens, D. (1993). Toward a better understanding of mixing in genetic algorithms. *Journal of The Society of Instrument and Control Engineers*, *32*(1), 10–16.

Goldberg, D. E., & Lingle, R. (1985). Alleles, loci, and the traveling salesman problem. *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, 154–159.

Goldberg, D. E., & 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., & Segrest, P. (1987). Finite Markov chain analysis of genetic algorithms. *Proceedings of the Second International Conference on Genetic Algorithms*, 1–8.

Grefenstette, J. J., Gopal, R., Rosmaita, B. J., & Van Gucht, D. (1985). Genetic algorithms for the traveling salesman problem. *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, 160–168.

Holland, J., Smith, D., & Oliver, T. (1985). *The cycle crossover: Genetic algorithms in epistatic domain.* Manuscript submitted for publication.

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

Holland, J. H., Holyoak, K. J., Nisbett, R. E., & Thagard, P. R. (1986). *Induction: Processes of inference, learning, and discovery.* Cambridge: MIT Press.

Horn, J., Goldberg, D. E., & Deb, K. (1992). *Research note: Long path problems for mutation-based algorithms* (IlliGAL Tech. Report No. 92011). Urbana-Champaign: University of Illinois.

Horn, J., & Nafpliotis, N. (1993). *Multiobjective optimization using the niched pareto genetic algorithm* (IlliGAL Report No. 93005). Urbana-Champaign: University of Illinois.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220(4598)*, 671–680.

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

Mahfoud, S., & Goldberg, D. E. (1992). *Parallel recombinative simulated annealing: A genetic algorithm* (IlliGAL Report No. 92002). Urbana-Champaign: University of Illinois.

Pettit, E., & Swigger, K. M. (1983). An analysis of genetic-based pattern tracking and cognitive-based component tracking models of adaptation. *Proceedings of the National Conference on Artificial Intelligence*, 327–332.

Rechenberg, I. (1973). *Evolutionstrategie* [Evolution strategy]. Stuttgart: Frommann–Holzboog.

Schaffer, J. D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, 93–100.

Schraudolph, N. N., & Belew, R. (1991). *Dynamic parameter encoding for genetic algorithms* (UCSD Tech. Report No. CS 90-175). La Jolla, CA: University of California, San Diego.

Smith, R. E. (1988). *An investigation of diploid genetic algorithms for adaptive search of nonstationary functions* (TCGA Report No. 88001). Tuscaloosa: The University of Alabama, The Clearinghouse for Genetic Algorithms.

Smith, R. E. (1993a). *Adaptively resizing populations: An algorithm and analysis* (TCGA Report No. 93001). Tuscaloosa: University of Alabama.

Smith, R. E. (1993b). Adaptively resizing populations: An algorithm and analysis. In *Proceedings of The Fifth International Conference on Genetic Algorithms*, 653, San Mateo, CA. Morgan Kaufman.

Smith, R. E., & Cribbs, H. B. (1994). *Is the learning classifier system a type of neural network?* Manuscript Submitted for Publication

Smith, R. E., Forrest, S., & Perelson, A. S. (1993). Searching for diverse, cooperative populations with genetic algorithms. *Evolutionary Computation*, *1*(2), 127–149.

Smith, R. E., & Goldberg, D. E. (1992). Diploidy and dominance in artificial genetic search. *Complex Systems*, *6*(3), 251–285.

Smith, R. E., & Gray, B. (1994). *Co-adaptive genetic algorithms: An example in othello strategy.* Manuscript submitted for publication.

Smith, R. E., & Huettner, K. (1994). *Hybridization of genetic algorithms and gradient search.* Unpublished manuscript.

Smith, R. E., & Ideh, G. (1994). *Adaptive population sizing and the royal road problems.* Unpublished Manuscript.

Spears, W. M. (1993). Crossover or mutation? In L. D. Whitley (Ed.), *Foundations of genetic algorithms II* (221–237). San Mateo, CA: Morgan Kaufmann.

Spears, W. M., & De Jong, K. A. (1991). An analysis of multi-point crossover. In G. Rawlins (Ed.), *Foundations of genetic algorithms* (301–315). San Mateo, CA: Morgan Kaufmann.

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

Whitley, D. (1989a). *The Genitor algorithm: Using genetic recombination to optimize neural networks* (Technical Report CS–89–107). Fort Collins: Colorado State University, Computer Science Department.

Whitley, D. (1989b). *Scheduling problems and optimal traveling salesmen: The genetic edge recombination operator* (Technical Report CS–89–106). Fort Collins: Colorado State University, Computer Science Department.