Genetic Algorithms

Kirkpatrick [1983] has described Simulated Annealing as "an example of an evolutionary process modeled accurately by purely stochastic means", but this is more literally true of another class of new optimization routines known collectively as Genetic Algorithms (GAs). These attempt to simulate the phenomenon of natural evolution first observed by Darwin [1859] and recently elaborated by Dawkins [1986].

In natural evolution each species searches for beneficial adaptations in an ever-changing environment. As species evolve these new attributes are encoded in the chromosomes of individual members. This information does change by random mutation, but the real driving force behind evolutionary development is the combination and exchange of chromosomal material during breeding.

Although sporadic attempts to incorporate these principles in optimization routines have been made since the early 1960s (see a review in Chapter 4 of Goldberg [1989]), GAs were first established on a sound theoretical basis by Holland [1975]. The two key axioms underlying this innovative work were that complicated nonbiological structures could be described by simple bit strings and that these structures could be improved by the application of simple transformations to these strings.

GAs differ from traditional optimization algorithms in four important respects:

- They work using an encoding of the control variables, rather than the variables themselves.
- They search from one population of solutions to another, rather than from individual to individual.
- They use only objective function information, not derivatives.
- They use probabilistic, not deterministic, transition rules.

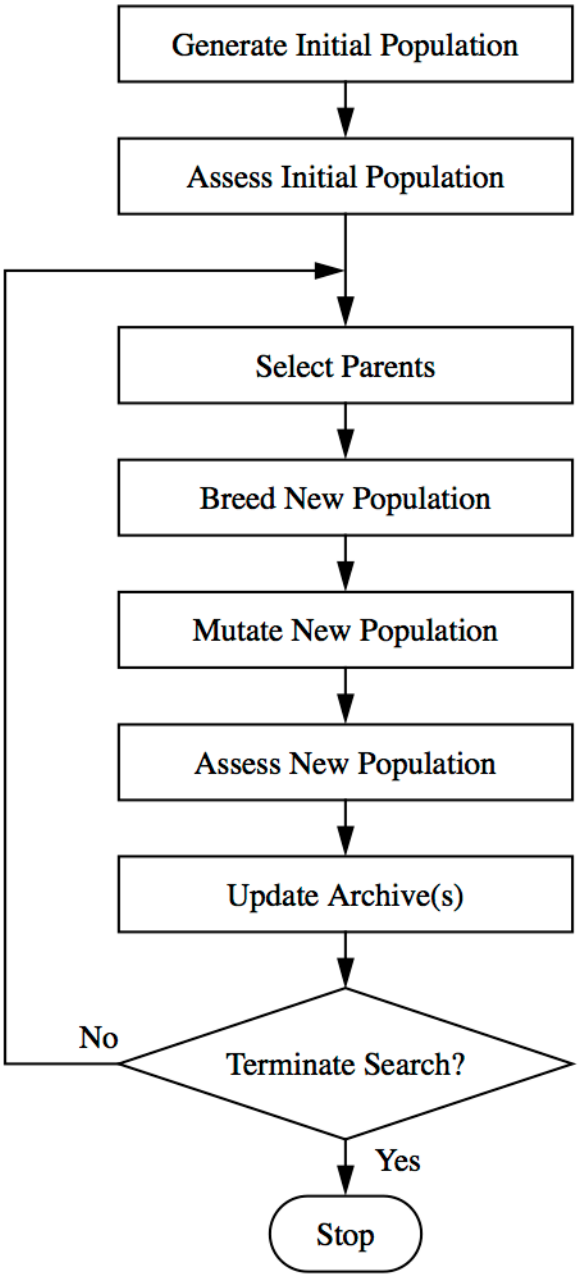
Of course, GAs share the last two attributes with SA and, not surprisingly, have found applications in many of the same areas.

The basic structure of a GA is shown in Figure 1. One minor change from the standard optimization routine flow diagram is the use of the word 'population' rather than 'solution'. A more major difference is that the usual operation of generating a new solution has been replaced by three separate activities — selection, recombination and mutation.

Solution Representation

In order to tackle a problem using a GA, candidate solutions must be encoded in a suitable form. In the traditional GA solutions are represented by binary bit strings. While integer control variables are easily encoded in this form, the representation of continuous control variables is not so simple. In general, the only option available is to approximate them (rescaled as

Figure 1: The Basic Structure of a Genetic Algorithm.



necessary) by equivalent integer variables. The accuracy with which an optimal solution can be resolved then depends on the encoded bit length of these integers, leading to an inevitable compromise between precision and execution time.[1]

For combinatorial optimization problems, problem-specific solution encodings, such as ordered lists, are necessary. For example, a solution to the TSP can represented by a string listing the cities in the order they are to be visited. Problem-specific operators are also required to manipulate such strings validly.

1. In fact, *Evolution Strategies* almost always perform better than Genetic Algorithms on optimization problems with continuous control variables and do not incur quantization errors, so there is little incentive to use a GA on such problems.

Parent Selection

The initial population for a GA search is usually selected randomly, although there may be occasions when knowledge-based selection is appropriate (Grefenstette [1987]). Within the algorithm, parent selection is based on the principle of 'survival of the fittest'. The standard procedure, known as *proportional selection*, is to define the probability of a particular solution i's selection to be:

$$P_{Si} = \frac{f_i}{f_\Sigma}\,, \tag{1}$$

if the objective function is to be maximized, where f_i is the *fitness* (objective value) of solution i, and

$$f_\Sigma = \sum_{i=1}^{N} f_i \tag{2}$$

is the total fitness of the population (of size N), or, if f is to be minimized,

$$P_{Si} = 1 - \frac{f_i}{f_\Sigma} \tag{3}$$

The parents for the new population are then selected by simulating the spinning of a suitably weighted roulette wheel N times.*

It is clear that f must always be positive for this scheme to be used. Its range and scaling are also important. For instance, early in a search it is possible for a few superindividuals (solutions with fitness values significantly better than average) to dominate the selection process. Various schemes have been suggested to overcome this potential danger, of which the simplest is *linear scaling*, whereby f is rescaled through an equation of the form:

$$\tilde{f} = af + b\,. \tag{4}$$

Coefficients a and b are chosen each generation so that the average values of f and $\tilde{f}$ are equal and so that the maximum value of $\tilde{f}$ is a specified multiple of (usually twice) the average. Linear scaling risks the introduction of negative values of $\tilde{f}$ for low performance solutions and must therefore be used with caution.

Baker [1985] suggested that P_{Si} should simply be made a (linear) function of the solution's ranking R_i within the population, so that

$$P_{Si} = \frac{S\,(N + 1 - 2R_i) + 2\,(R_i - 1)}{N\,(N-1)}\,, \tag{5}$$

where S is a parameter known as the *selection pressure*. Thus, the best solution has a selection probability of S/N and the worst one of $(2-S)/N$. If $S = 1$ each solution has an equal selection probability; if $S = 2$ the worst solution has a zero probability of selection, and the best has

* In the latter case the resulting probabilities must first be renormalized so that they sum to 1.

a probability twice the average. This ranking scheme has been found to overcome the problems of over- or underselection, without revealing any obvious drawbacks.

Roulette wheel selection suffers from the disadvantage of being a high-variance process with the result that there are often large differences between the actual and expected numbers of individual solutions selected — there is no guarantee that the best solution will be selected. De Jong [1975] tested an *elitist* scheme, which gave just such a guarantee by enlarging the set of selected parents to include a copy of the best solution if it hadn't been chosen. He found that on problems with just one maximum (or minimum) the algorithm performance was much improved, but on multimodal problems it was degraded.

Numerous schemes which introduce various levels of determinism into the selection process have been investigated. Overall, it seems that a procedure entitled *stochastic remainder selection without replacement* offers the best performance. In this, the expected number of copies of each solution is calculated as

$$E_i = N P_{Si} \,. \tag{6}$$

Each solution is then selected I_i times, I_i being the integer part of E_i. The fractional remainder

$$R_i = E_i - I_i \tag{7}$$

is treated as the probability of further selection. For example, a solution for which $E_i = 1.8$ would certainly be selected once and would be selected again with probability 0.8. Each solution is successively subjected to an appropriately weighted simulated coin toss until sufficient parents have been selected.

Another popular selection method (due to Goldberg and Deb [1991]) is *tournament selection*, in which a small subset of solutions is chosen randomly. The best one or two solutions in this subset are then selected for mating. Like ranking selection, tournament selection is less susceptible to takeover by good individuals, and the selection pressure can be adjusted by controlling the size of the subset used.

In a *generational* GA the entire population is replaced by newly selected solutions before the recombination, mutation and assessment phases of the algorithm. In a *steady-state* GA two parents are selected from the current population in order to produce one or two offspring which then replace one or two members of the current population (either the worst, or randomly selected members, or their parents). Variations between these two extremes in which a fraction of the population is replaced each 'generation' are also possible, of course.

It is clear from the foregoing discussion that there are many possible selection schemes which can be implemented in GAs. Unfortunately it is found that different schemes perform better on different problems, so it is difficult to give general advice as to which to choose.

Population Recombination (Breeding)

Whatever selection procedure is used, it is does not, of course, introduce any new solutions. Solutions are selected in order to serve as progenitors (breeding stock) for a new generation. It is in the recombination (breeding) phase that the algorithm attempts to create new, improved solutions. The key procedure is that of *crossover*, in which the GA seeks to construct better solutions by combining the features of good existing ones.

The simplest form of crossover (*one point crossover*) proceeds as follows. First, the selected individuals are paired off at random to give sets of potential parents. Second, pairs of solutions are chosen to undergo crossover with probability P_C. If the simulated weighted coin toss rejects crossover for a pair, then both solutions remain in the population unchanged. However, if it is approved, then two new solutions are created by exchanging all the bits following a randomly selected locus on the strings. For example, if crossover after position 5 is proposed between solutions

1 0 0 1 1 1 0 1

and

1 1 1 1 0 0 0 0 ,

the resulting offspring are

1 0 0 1 1 0 0 0

and

1 1 1 1 0 1 0 1 .

A slightly more complex operator, first proposed by Cavicchio [1970], is *two point crossover* in which two crossover points are randomly selected and the substrings between (and including) those positions are exchanged. If necessary, strings are treated as continuous rings. Thus, if crossover between points 6 (selected first) and 2 is proposed for strings

1 0 0 1 1 1 0 1

and

1 1 1 1 0 0 0 0 ,

the resulting offspring are

1 1 0 1 1 0 0 0

and

1 0 1 1 0 1 0 1 .

Whereas, if crossover is between points 2 (selected first) and 6, the offspring are

1 1 1 1 0 0 0 1

and

1 0 0 1 1 1 0 0 .

The even-handedness of two point crossover is appealing. There is no intuitive reason why right-hand bits should be exchanged more frequently than left-hand ones, unless the string represents the binary coding of a single integer or continuous variable, in which case, of course, the significance of individual bits decreases from left to right.

De Jong [1975] tested multiple point crossover operators, which exchange more than one substring, and found that performance is degraded as more substrings are swapped. Although it is essential to introduce some changes in order to make progress, too many alterations make the probability of destroying the good features of a solution unacceptably high. An effective GA search requires a balance between *exploitation* (of good features in existing solutions) and *exploration* — introducing new (combinations of) features.

The performance of the recombination phase of a GA can also be improved by requiring that crossover introduce variation whenever possible. For instance, if the strings

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

and

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

are chosen partners, then only exchanges involving bits 2, 3 or 4 will result in offspring differing from their parents. Booker [1987] suggested that a general solution to this problem is to perform crossover between points in the parents' *reduced surrogates*, which contain just the nonmatching bits.

For many real applications, problem-specific solution representations and crossover operators have been developed. This flexibility is one of the attractions of GAs — it is very easy to introduce heuristic (knowledge-based) operators, which can substantially improve algorithm performance. This topic was very well reviewed by Davis [1991].

Population Mutation

Although *mutation* merits a separate box in the flow diagram, it is very much a background operator in most GA applications (as it is in nature). The purpose of the mutation stage is to provide insurance against the irrevocable loss of genetic information and hence to maintain diversity within the population. For instance, if every solution in the population has 0 as the value of a particular bit, then no amount of crossover will produce a solution with a 1 there instead.

In traditional GAs every bit of every solution is potentially susceptible to mutation. Each bit is subjected to a simulated weighted coin toss with a probability of mutation P_M, which is usually very low (of the order of 0.01 or less). If mutation is approved, the bit changes value (in the case of a binary coding from 0 to 1 or 1 to 0).

There are schools of thought in the GA community which believe that mutation should only

take place in solutions for which crossover was not approved or that only one mutation per solution should occur. Undoubtedly there are classes of problem for which each scheme is the most effective.

Initial Population Generation

The initial population which is then evolved by the GA should, if possible, be well spread through the search space, so that it is well sampled. There are two standard ways of achieving this:

- Generate the initial population entirely by random sampling.
- Generate the initial population by repeated mutation of a given initial solution. As the mutation operator makes small changes it is advisable to make multiple mutations to ensure "genetic diversity" in the population. A possible initialisation routine might, for instance, take the initial solution (population member 1), make 10 mutations and save this solution (population member 2), make 10 further mutations and save this solution (population member 3) etc.

If you have prior knowledge of the problem you are solving, the GA's performance can often be enhanced by "seeding" the initial population with one (or more, as long as they are diverse) good solutions.

Population Assessment

Like the SA algorithm, a GA does not use derivative information, it just needs to be supplied with a fitness value for each member of each population. Thus, the evaluation of the problem functions is essentially a 'black box' operation as far as the GA is concerned.

The guidelines for constraint handling in a GA are basically identical to those outlined for the SA algorithm. As long as there are no equality constraints, the problem is not very heavily constrained, and the feasible space is not disjoint, then infeasible solutions can simply be 'rejected'. In a GA this means ensuring that those particular solutions are not selected as parents in the next generation, e.g. by allocating them a zero survival probability.

If these conditions on the constraints are not met, then a penalty function method should be used. A suitable form for a GA is:

$$f_A(\mathbf{x}) = f(\mathbf{x}) + M^k \mathbf{w}^T \mathbf{c}_V(\mathbf{x}) , \tag{8}$$

where $\mathbf{w}$ is a vector of nonnegative weighting coefficients, the vector $\mathbf{c}_V$ quantifies the magnitudes of any constraint violations, M is the number of the current generation and k is a suitable exponent. The dependence of the penalty on generation number biases the search increasingly heavily towards feasible space as it progresses.

Search Termination

In some simple GA implementations the search is terminated after a fixed number of generations, with the number being a function of the size of the problem. Alternatively, the search can be halted when it ceases to make progress. Lack of progress in a GA can be defined in a number of ways, but a useful basic definition is

- no new best solution being found for, say, three consecutive generations.

Control Parameters

The efficiency of a GA is highly dependent on the values of the algorithm's control parameters. Assuming that basic features like the selection procedure are predetermined, the control parameters available for adjustment are:

- the population size N,
- the crossover probability P_C, and
- the mutation probability P_M.

De Jong [1975] made some recommendations based on his observations of the performance of GAs on a test bed of 5 problems, which included examples with difficult characteristics such as discontinuities, high dimensionality, noise and multimodality. His work suggested good settings to be

$$(N, P_C, P_M) = (50, 0.60, 0.001) .$$

Grefenstette [1986] went one stage further and used a GA to optimize these parameters for a test bed of problems. He concluded that

$$(N, P_C, P_M) = (30, 0.95, 0.010)$$

resulted in the best performance when the average fitness of each generation was used as the indicator, while

$$(N, P_C, P_M) = (80, 0.45, 0.010)$$

gave rise to the best performance when the fitness of the best individual member in each generation was monitored. The latter is, of course, the more usual performance measure for optimization routines.

In general, the population size should be no smaller than 25 or 30 whatever the problem being tackled, and for problems of high dimensionality larger populations (of the order of hundreds) are appropriate.

Computational Considerations

Unlike SA, which is intrinsically a sequential algorithm, GAs are particularly well-suited to implementation on parallel computers or multiprocessor networks. Evaluation of the objective

function and constraints can be done simultaneously for a whole population, as can the production of the new population by mutation and crossover. Thus, on a highly parallel machine/ network, a GA can be expected to run nearly N times as fast for many problems, where N is the smaller of the population size and the number of processors.

Example: Rosenbrock's Function: $f(\mathbf{x}) = 100\,(x_2 - x_1^2)^2 + (1 - x_1)^2$

Figure 2 shows the progress of a GA on Rosenbrock's function. This is presented purely for purposes of comparison. Each member of the 1st, 10th, 20th and 30th generations is shown (by the • symbol). The convergence of the population (of size 50) to the neighbourhood of the optimum at $(1, 1)$ is readily apparent. 30 bit binary chromosomes were used to represent the

Figure 2: Minimization of Rosenbrock's Function by a GA — Population Distributions.

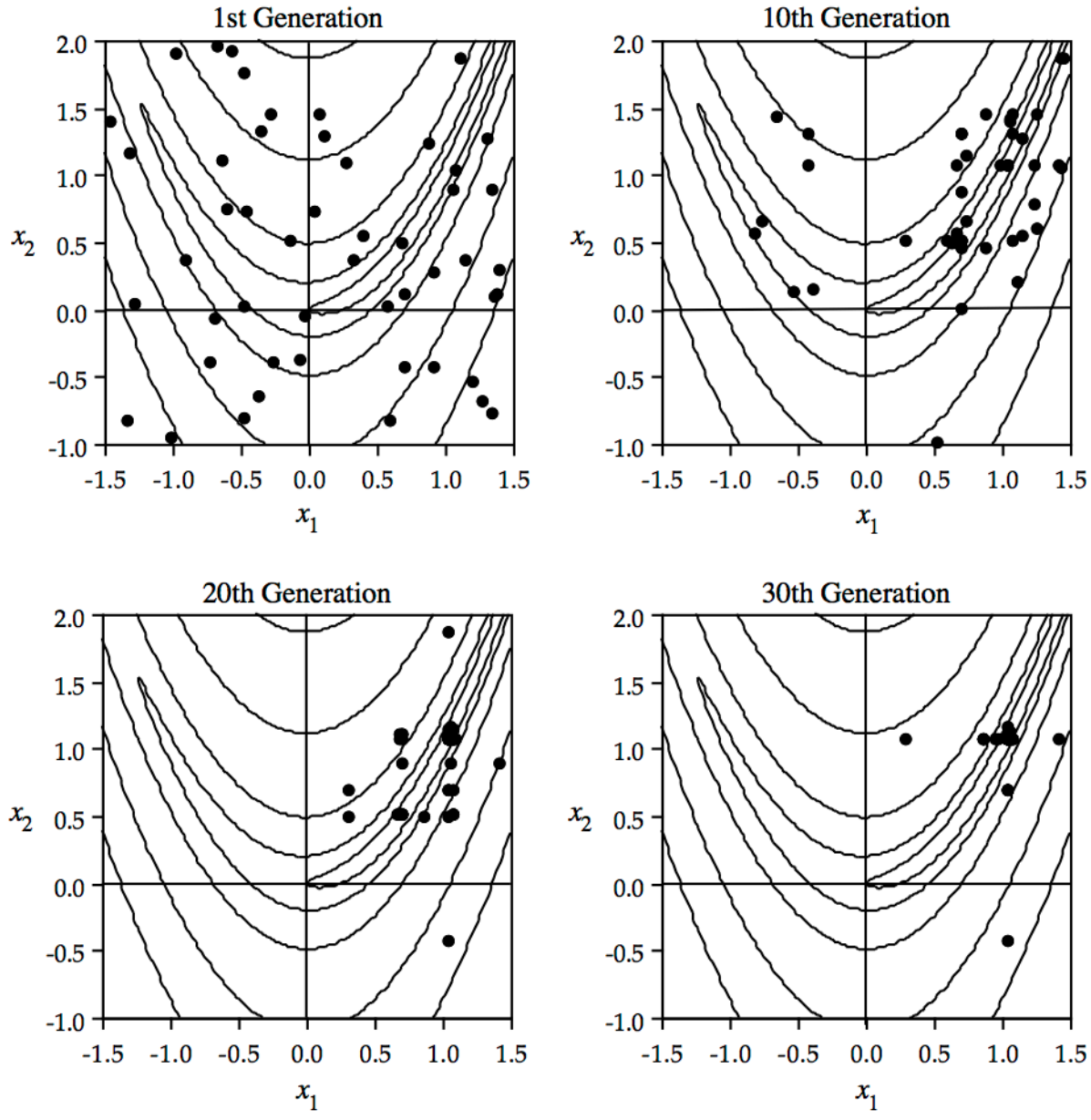
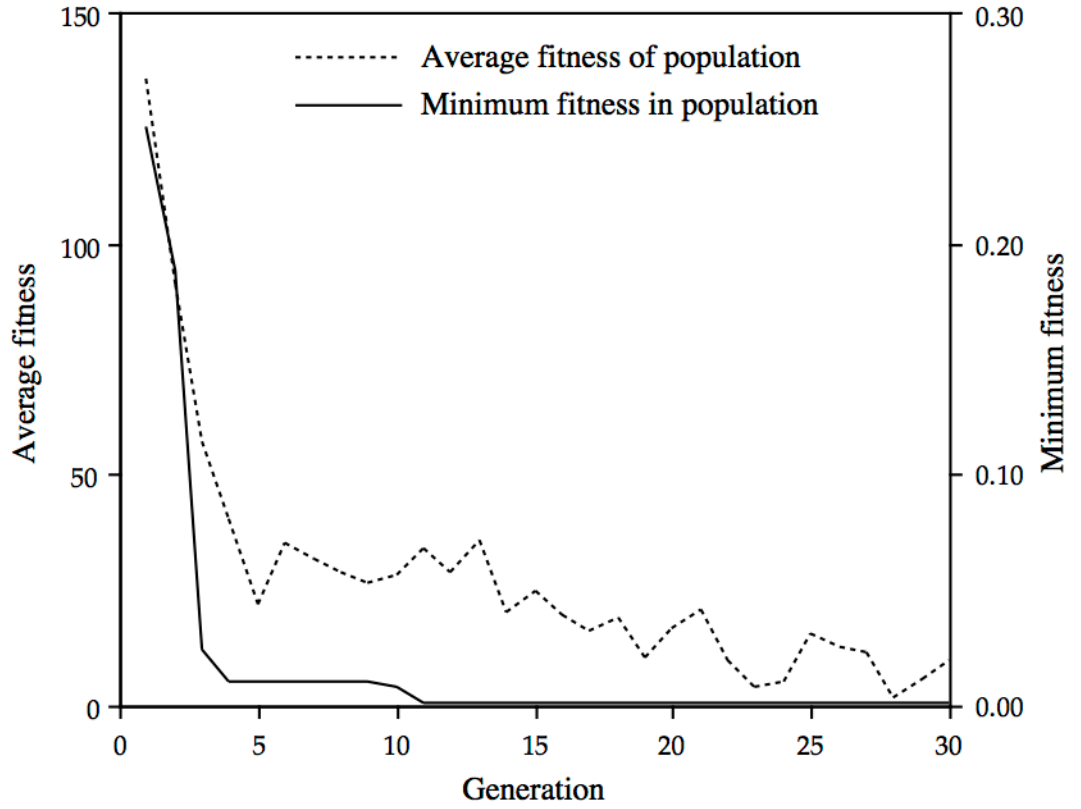


Figure 3: Minimization of Rosenbrock's Function by a GA — Objective Reduction.



control variables, giving quantization errors of just 1.4×10^{-9}.

Figure 3 shows the progress in reducing the objective function for the same search. Both the fitness of the best individual within each population and the population average fitness are shown (note that the scales are different). These are the two standard measures of progress in a GA run. The difference between these two measures is indicative of the degree of convergence in the population.

In this particular example, the GA has been successful in locating the global optimum, but it must be noted that GAs are often found to experience convergence difficulties. In many applications GAs locate the neighbourhood of the global optimum extremely efficiently but have problems converging onto the optimum itself. In such instances a hybrid algorithm, for example using a GA initially and then switching to a low temperature SA search, can prove effective.

References

Baker, J.E., (1985) "Adaptive Selection Methods for Genetic Algorithms", 101-111 in *Proceedings of an International Conference on Genetic Algorithms and their Applications* (J.J. Grefenstette, editor), Lawrence Erlbaum Associates, Hillsdale, NJ.

Booker, L.B., (1987) "Improving Search in Genetic Algorithms", 61-73 in *Genetic Algorithms and Simulated Annealing* (L. Davis, editor), Pitman, London.

Cavicchio, D.J., (1970) *Adaptive Search Using Simulated Evolution*, Ph.D. Thesis, University of Michigan, Ann Arbor, MI.

Darwin, C., (1859) *On The Origin of Species*, 1st edition (facsimile – 1964), Harvard University Press, Cambridge, MA.

Davis, L., (1991) *Handbook of Genetic Algorithms*,Van Nostrand Reinhold, New York, NY.

Dawkins, R., (1986) *The Blind Watchmaker*, Penguin, London.

De Jong, K.A., (1975) *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph.D. Thesis, University of Michigan, Ann Arbor, MI.

Goldberg, D.E., (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, Reading, MA.

Goldberg, D.E., and K. Deb (1991) "A Comparative Analysis of Selection Schemes used in Genetic Algorithms", 69-93 in *Foundations of Genetic Algorithms* (G.J.E. Rawlins, editor), Morgan Kaufmann, San Mateo, CA.

Grefenstette, J.J., (1986) "Optimization of Control Parameters for Genetic Algorithms", *IEEE Trans. Syst., Man, Cyber.* **SMC-16**, 122-128.

Grefenstette, J.J., (1987) "Incorporating Problem Specific Knowledge into Genetic Algorithms", 42-60 in *Genetic Algorithms and Simulated Annealing* (L. Davis, editor), Pitman, London.

Holland, J.H., (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI.

Hollstien, R.B., (1971) *Artificial Genetic Adaptation in Computer Control Systems*, Ph.D. Thesis, University of Michigan, Ann Arbor, MI.

Kirkpatrick, S., C.D. Gerlatt, Jr., and M.P. Vecchi (1983) "Optimization by Simulated Annealing", *Science* **220**, 671-680.