# Evolutionary Algorithms

**Zbigniew Michalewicz**
*University of North Carolina, Charlotte*

**Marc Schoenauer**
*Ecole Polytechnique*



## GLOSSARY

**fitness** The measure of adaptation of the individuals to their artificial environment—basis for Darwinian selection mechanisms.

**genotype** The representation of an individual that will be transmitted (after possible modification by variation operators) to its offspring during evolution.

**hybridization** Use of existing (nonevolutionary) optimization techniques within an evolutionary algorithm.

**individual** Possible solution to the problem at hand, that is, from a mathematical point of view, a point of the search space.

**phenotype** The behavioral part of an individual. The phenotype is computed, or "decoded," from the genotype, and the fitness is computed on the phenotype.

**population** Set of individuals. The population is generally of fixed size.

**replacement** Second phase of artificial Darwinism. The fittest (deterministically or stochastically) individuals will survive.

**representation** Synonym for the genotypic space (space of genotypes). The choice of a representation for a given problem is the very first step of the design of an evolutionary algorithm.

**selection** First phase of artificial Darwinism. The fittest (deterministically or stochastically) individuals will reproduce.

**variation operators** Modification of the individuals in the search space. According to Darwin's principles, variation operators are not aware of the fitness of the individuals.

**EVOLUTIONARY COMPUTING** is an exciting development in computing science. It amounts to building, applying, and studying algorithms based on the Darwinian principles of natural selection ("survival of the fittest") and undirected variations. Evolutionary algorithms can also be viewed as an interesting category of modern heuristic search. This overview article presents the main paradigms of evolutionary algorithms (genetic algorithms, evolution strategies, evolutionary programming, genetic programming) as well as the trend for unification of these paradigms and hybridization with other existing search techniques. A brief survey of theoretical results is presented, as well as a list of application areas ranging from optimization, modeling, and simulation to entertainment.

## I. INTRODUCTION

The evolutionary computation (EC) techniques are stochastic algorithms whose search methods model some natural phenomena: genetic inheritance and Darwinian strife for survival. The idea behind evolutionary algorithms is to do what nature does. Let us

take rabbits as an example: At any given time there is a population of rabbits. Some of them are faster and smarter than other rabbits. These faster, smarter rabbits are less likely to be eaten by foxes, and therefore more of them survive to do what rabbits do best: make more rabbits. Of course, some of the slower, dumber rabbits will survive just because they are lucky. This surviving population of rabbits starts breeding. The breeding results in a good mixture of rabbit genetic material: some slow rabbits breed with fast rabbits, some fast with fast, some smart rabbits with dumb rabbits, and so on. And on the top of that, nature throws in a "wild hare" every once in a while by mutating some of the rabbit genetic material. The resulting baby rabbits will (on average) be faster and smarter than these in the original population because more faster, smarter parents survived the foxes. (It is a good thing that the foxes are undergoing a similar process—otherwise the rabbits might become too fast and smart for the foxes to catch any of them). So the metaphor underlying evolutionary algorithms is that of natural evolution. In evolution, the problem each species faces is one of searching for beneficial adaptations to a complicated and changing environment. The "knowledge" that each species has gained is embodied in the makeup of the chromosomes of its members. From the point of view of optimization, EC is a powerful stochastic zeroth-order method (i.e., requiring only values of the function to optimize) that can find the global optimum of very rough functions. This allows EC to tackle optimization problems for which standard optimization methods (e.g., gradient-based algorithms requiring the existence and computation of derivatives) are not applicable. Moreover, most traditional methods are local in scope, thus they identify only the local optimum closest to their starting point.

## II. AN ALGORITHM

For the sake of clarity, we shall try to introduce a general framework that can account as much as possible for most of the existing evolutionary algorithms.

Let the search space be a metric space $E$, and let $F$ be a function $E \to \Re$ called the *objective* function. The problem of evolutionary optimization is to find the maximum of $F$ on $E$ (the case of minimization is easily handled by considering $-F$).

A *population* of size $P \in \Re$ is a set of $P$ *individuals* (points of $E$) not necessarily distinct. This population is generally initialized randomly (at time $t = 0$) and uniformly on $E$. The *fitnesses* of all individuals are computed (on the basis of the values of the objective function); a fitness value is represented as a positive real number—the higher the number, the better the individual. The population then undergoes a succession of *generations;* the process is illustrated in Fig. 1.

Several aspects of the evolutionary procedure require additional comments:

- *Statistics and stopping criterion:* The simplest stopping criterion is based on the generation counter $t$ (or on the number of function evaluations). However, it is possible to use more complex stopping criteria, which depend either on the evolution of the best fitness in the population along generations (i.e., measurements of the gradient of the gains over some number of generations) or on some measure of the diversity of the population.
- *Selection:* Choice of some individuals that will generate offspring. Numerous selection processes can be used, either deterministic or stochastic. All are based on the fitness of the individuals. Depending on the selection scheme used, some individuals can be selected more than once. At that point, selected individuals give birth to copies of themselves (clones).
- *Application of variation operators:* To each one of these copies some operator(s) are applied, giving birth to one or more offspring. The choice among possible operators is stochastic, according to user-supplied probabilities. These operators are always stochastic operators, and one usually distinguishes between *crossover* (or *recombination*) and *mutation* operators:
  - Crossover operators are operators from $E^k$ into $E$, i.e., some parents exchange genetic material to build up one offspring. In most cases, crossover involves just two parents ($k = 2$), however, it need not be the case. In a recent

```
procedure evolutionary algorithm
begin
   t ← 0
   initialize population
   evaluate population
   while (not termination-condition do
   begin
      t ← t + 1
      select individuals for reproduction
      apply variation operators
      evaluate newborn offspring
      replace some parents by some offspring
   end
end
```

**Figure 1** The structure of an evolutionary algorithm.

study, the authors investigated the merits of "orgies," where more than two parents are involved in the reproduction process. Evolution strategies and scatter search techniques also proposed the use of multiple parents.
  - Mutation operators are stochastic operators from $E$ into $E$.
- *Evaluation:* Computation of the fitnesses of all newborn offspring. As mentioned earlier, the fitness measure of an individual is directly related to its objective function value.
- *Replacement:* Choice of which individuals will be part of the next generation. The choice can be made either from the set of offspring only (in which case all parents "die") or from both sets of offspring and parents. In either case, this replacement procedure can be deterministic or stochastic.

Sometimes the variation operators are defined on the same space as the objective function (called *phenotype space* or behavioral space); in other cases, an intermediate space is introduced (called *genotype space* or representation space). The mapping from the phenotype space in the genotype space is termed *coding*. The inverse mapping from the genotype space in the phenotype space is termed *decoding*. Genotypes undergo variation operators, and their fitness is evaluated on the corresponding phenotype. The properties of the coding mappings can greatly modify the global behavior of the evolutionary algorithm.

## III. GENETIC ALGORITHMS

In the canonical genetic algorithm (GA), the genotype space is $\{0,1\}^n$. Note that the phenotype space can be any space, as long as it can be coded into bit string genotypes. The selection scheme is proportional selection (the best-known being the *roulette wheel selection*): $P$ random choices are made in the whole population, each individual having a probability proportional to its fitness of being selected. The crossover operators replace a segment of bits in the first parent string by the corresponding segment of bits from the second parent, and the mutation operator randomly flips the bits of the parent according to a fixed user-supplied probability. In the replacement phase, all $P$ offspring replace all parents. Due to that generational replacement, the best fitness in the population can decrease: The original GA strategy is not *elitist*.

In more recent works, the genotype space can be almost any space, as long as some crossover and mutation operators are provided. Moreover, proportional selection has been gradually replaced by ranking selection (the selection is performed on the rank of the individuals rather than on their actual fitness) or tournament selection (one selects the best individual among a uniform choice of $T$ individuals, $T$ ranging from 2 to 10). Finally, most users use the elitist variant of replacement, in which the best individual of generation $t$ is included in generation $t + 1$, whenever the best fitness value in the population decreases.

## IV. EVOLUTION STRATEGIES

The original evolution strategy (ES) algorithm handles a "population" made of a single individual given as a real-valued vector. This individual undergoes a Gaussian mutation: addition of zero-mean Gaussian variable of standard deviation $\sigma$. The fittest individual from the parent and the offspring becomes the parent of the next generation. The critical feature is the choice of parameter $\sigma$: Originally, the so-called 1/5 thumb rule [i.e., When more than 1/5 mutations are successful (respectively, unsuccessful), increase (respectively, decrease) $\sigma$] was used to adjust parameter $\sigma$ along evolution.

More recent ES algorithms are population-based algorithms, termed $(\mu,\lambda) - \text{ES}$ or $(\mu + \lambda) - \text{ES}$: $\mu$ parents generate $\lambda$ offspring. (There is no selection at that level, i.e., every parent produces $\lambda/\mu$ offspring on average.)

The main operator remains mutation. When working on real-valued vectors (still their favorite universe) ESs generally use the powerful paradigm of *self-adaptive mutation:* The standard deviations of Gaussian mutations are part of the individuals, and undergo mutation as well. Last, ESs now frequently use a global recombination operator involving all individuals in the population.

The replacement step is deterministic, i.e., the best $\mu$ individuals become the parents of the next generation, chosen among the $\mu + \lambda$ parents plus offspring in the elitist $(\mu + \lambda) - \text{ES}$ scheme, or among the $\lambda$ offspring in the nonelitist $(\mu,\lambda) - \text{ES}$ scheme (with $\lambda \geq \mu$). Typical values for $(\mu,\lambda)$ are (1,7), (10,100) or (30,200).

## V. EVOLUTIONARY PROGRAMMING

Originally designed to evolve finite state machines, evolutionary programming (EP) emphasizes the phenotype space. As in ESs, there is no initial selection: Every

individual in the population generates one offspring. Moreover, the only evolution operator is mutation. Finally, the best $P$ individuals among parents and offspring become the parents of the next generation.

Recent advances handle any space, still emphasize the use of mutation as the only operator, independently design the self-adaptive Gaussian deviations for real-valued variables, and now use a stochastic tournament replacement scheme: Each individual (among the $2P$ parents plus offspring) encounters $T$ random opponents, increasing its score by one point if it has better fitness. The $P$ individuals having the highest scores get along to the next generation. Note that EP replacement scheme is always *elitist*.

## VI. GENETIC PROGRAMMING

Genetic programming as a method for evolving computer programs first appeared as an application of GAs to tree-like structures. Original GP evolves tree structures representing LISP-like S expressions. This allows us to define very easily a closed crossover operator (by swapping subtrees between two valid S expressions, we always gets a valid S expression). The usual evolution scheme is the steady-state genetic algorithm (SSGA): A parent is selected by tournament (of size 2 to 7 typically) and generates an offspring by crossover only (the other parent is selected by a tournament of usually smaller size). The offspring is then put back in the population using a death-tournament: $T$ individuals are uniformly chosen, and the one with the worse fitness gets replaced by the newborn offspring.

More recently, mutation operators, for example, random replacement of a subtree or random change of a node or a leaf, have been used—see the state-of-the-art books listed in the Bibliography.

## VII. MODERN TRENDS: HYBRID METHODS

Many researchers modified further evolutionary algorithms by "adding" some problem-specific knowledge to the algorithm. Several papers have discussed initialization techniques, different representations, decoding techniques (mapping from genetic representations to phenotypic representations), and the use of heuristics for variation operators. Davis wrote (in the context of classical, binary GAs):

> It has seemed true to me for some time that we cannot handle most real-world problems with binary representations and an operator set consisting only of binary crossover and binary mutation. One reason for this is that nearly every real-world domain has associated domain knowledge that is of use when one is considering a transformation of a solution in the domain. . . . I believe that genetic algorithms are the appropriate algorithms to use in a great many real-world applications. I also believe that one should incorporate real-world knowledge in one's algorithm by adding it to one's decoder or by expanding one's operator set.

Such hybrid/nonstandard systems enjoy a significant popularity in evolutionary computation community. Very often these systems, extended by the problem-specific knowledge, outperform other classical evolutionary methods as well as other standard techniques. For example, a system called Genetic-2N, constructed for the nonlinear transportation problem, used a matrix representation for its chromosomes, a problem-specific mutation (main operator, used with probability 0.4), and arithmetical crossover (background operator, used with probability 0.05). It is hard to classify this system; it is not really a genetic algorithm, because it can run with a mutation operator only without any significant decrease of quality of results. Moreover, all matrix entries are floating-point numbers. It is not an evolution strategy, because it does not use Gaussian mutation, nor does it encode any control parameters in its chromosomal structures. Clearly, it has nothing to do with genetic programming and very little (matrix representation) with evolutionary programming approaches. It is just an evolutionary computation technique aimed at particular problems.

## VIII. COMPARISON

Many papers have been written on the similarities and differences between these approaches. Clearly, different points of view can be adopted.

- *The representation issue:* Original EP, ESs, and GAs address only finite state machines, real numbers and bit strings, respectively. However, recent tendencies indicate that this is not a major difference. More important is the adequacy of the variation operators to the chosen representation and the objective function (i.e., the fitness landscape).
- *Bottom-up versus top-down, and the usefulness of crossover:* According to the schema theorem of Holland and Goldberg, GA's main strength comes from the crossover operator: Better and better solutions are built

by exchanging *building blocks* from partially good solutions previously built, in a bottom-up approach. The mutation operator is then considered as a background operator. On the other hand, the philosophy behind EP and ESs is that such building blocks might not exist, at least for most real-world problems. This top-down view considers selective pressure plus genotypic variability brought by mutation to be sufficient.

The discussion on crossover has been going on for a long time. And even when crossover was experimentally demonstrated beneficial to evolution, it could be because it acts like a large mutation; recent experiments suggest that the answer is highly problem dependent.

Yet another example of the duality between crossover and mutation comes from GP history: the original GP algorithm used only crossover, with no mutation at all, the very large population size being assumed to provide all the necessary building blocks to represent at least one sufficiently good solution. But more recent works on GP accommodate mutation also, on a much smaller population.

- *Mutation operators:* The way in which mutation operators are applied differs from one algorithm to another.

GA uses a static mutation rate or user-prescribed evolution scheme to globally adjust either the mutation rate (i.e., the number of individuals that undergo mutation) or the strength of mutation (i.e., the average number of bits that are flipped in an individual undergoing mutation).

Originally, ES used a heuristic adaptation mechanism (the 1/5 rule), which was later turned into the modern self-adaptive mutation: All individuals carry their own copy of the standard deviation(s) of the mutation. These variances undergo in turn mutation, and the individual is further modified according to the new value of the variance, which is therefore evolved and optimized "for free." The strength of mutation in EP is historically defined as a function of the relative fitness of the individual at hand before independently turning to self-adaptation.

Note that self-adaptive mutation rates (i.e., dependent on the individual) have a significant impact only when all individuals undergo mutation, which is not true for GAs where the mutation rate is generally low. However, the importance of local mutation is confirmed by theoretical results in ESs. A prerequisite for convergence is the *strong causality principle* emphasized by Rechenberg of ESs: Small mutations should have small effects on the fitness. This is not the case when floating-point numbers are encoded into binary strings (as is the case for classical GAs).

- The selection–replacement mechanisms range from the totally stochastic fitness proportional selection of GAs with generational replacement, to the deterministic $(\mu, \lambda)$ replacement of ES, through the stochastic, but elitist, tournament replacement of EP and the steady-state scheme (tournament selection and death tournament replacement) used in GP. Though some studies have been devoted to selection/replacement mechanisms, the choice of a selection scheme for a given problem (fitness-representation-operators) is still an open question (and is probably problem dependent).

The current trend in the EC community is to mix up all of these features to best fit the application at hand, on a few pragmatic bases: Some ESs applications deal with discrete or mixed real-integer spaces, the "binary is the best" credo of GAs has been successfully attacked (Antonisse, 1989), and the schema theorem extended to any representation. Note that some ES variations incorporate crossover, mutation has been added to GP, and so on. And the different selection operators are more and more being used now by the whole community.

On the other hand, such hybrid algorithms, by getting away from the simple original algorithms, also escape the few available theoretical results. Thus, the study of the actual complexity of the resulting algorithms remains unreachable.

## IX. THEORETICAL RESULTS

Theoretical studies of evolutionary algorithms are of two types: An evolutionary algorithm can be viewed as a Markov chain in the space of populations, because population at time $t + 1$ only depends on population at time $t$ (at least in the standard algorithms). The full theory of Markov chains can then be applied. On the other hand, the specific nature of evolution strategies allowed precise theoretical studies on the rate of convergence of these algorithms using probability calculus (at least for locally convex functions).

Results based on Markov chains analysis are available for the standard GA scheme (proportional selection with fixed mutation rate). The need for an elitist strategy is emphasized by Rudolph. When the mutation rate is allowed to decrease along generations, techniques borrowed from the field of simulated annealing give more precise convergence results in probability. Yet a different approach is used by Cerf that considers the GA as a stochastic perturbation of

a dynamical system (a caricature GA). The powerful Friedlin-Wentzell theory can then be applied, resulting in a lower bound on the population size for a convergence in finite time of a modified GA (in which the selection strength and mutation rate are carefully modified along generations). However, even this last result is nonconstructive, i.e., of limited use when actually designing an instance of evolutionary algorithm for a particular problem.

On the other hand, ESs have considered theoretical studies from the very beginning: studies on the sphere and corridor models gave birth to the 1/5 rule, with determination of the optimal update coefficients for the mutation rate. The theory of ESs later developed to consider global convergence results in probability for the elitist models, as well as for the nonelitist $(1, \lambda) - \mathrm{ES}$. The whole body of work by Beyer concentrates on the optimal progress rate for different variants of evolution strategies (and, for instance, justify some parameter settings for self-adaptive mutation given by Schwefel). The main weakness of these results remains that they were derived on simple models of function; their main results (e.g., optimal parameter settings) are nevertheless applied without further justification to any function—and usually prove to be efficient hints.

However, one should keep in mind that all of the above theoretical analyses address some simple models of evolutionary algorithms. As stated earlier, the modern trends of EC gave birth to hybrid algorithms, for which generally no theory is applicable.

## X. APPLICATION AREAS

Although it is often stressed that an evolutionary algorithm is not an optimizer in the strict sense, optimization problems form the most important application area of EAs. Some conferences dedicated to application of EAS and their proceedings provide a wide overview of actual applications. Another regularly updated source is the Evonet *Evolution@work* database.

This section will survey the preferred domains of application of EAs. The different subdomains are distinguished according to the type of search space they involve.

### A. Discrete Search Spaces

Hard combinatorial optimization problems (NP-hard, NP-complete) involve huge discrete search spaces, and have been studied extensively by the operational research community. Two different situations should be considered: academic benchmark problems and large real-world problems.

As far as benchmark problems are concerned, it is now commonly acknowledged that EAs alone cannot compete with OR methods. However, recent advances in hybrid algorithms termed *Genetic local search,* where the EA searches the space of local optima with respect to some OR heuristic, have obtained the best results so far on a number of such benchmark problems.

The situation is slightly different for real-world problems: "Pure" OR heuristics generally do not directly apply, and OR methods have to take into account problem specificities. This is true of course for EAs, and there are many success stories where EAs, carefully tuned to the problem at hand, have been very successful, for instance, in the broad area of scheduling.

### B. Parametric Optimization

The optimization of functions with floating-point variables has been thoroughly studied by practitioners, and many very powerful methods exist. Though the most well-known address linear or convex problems, many other cases can be handled successfully. Hence the niche for EAs is quite limited in that area, and only highly multimodal and irregular functions should be considered for EAs. However, successes have been encountered in such situations, in different domains ranging from electromagnetism to control and to fluid dynamics.

The situation drastically changes, however, when dealing with multiobjective problems: Evolutionary multi-objective (EMO) algorithms are the only ones that can produce a set of best possible compromise (the *Pareto set*) and have recently received increased attention. EMO algorithms use the same variation operators as standard EAs, but the Darwinian components are modified to take into account the multivalued fitness.

### C. Mixed Search Spaces

When it comes to mixed search spaces, that is, when different types of variables are involved (generally both continuous and discrete variables), almost no classical optimization method applies, although some OR methods can be used if continuous variables are transformed into intervals, or continuous methods can be applied to the discrete variables. All of these

approaches can easily fall into traps due to the very different nature of continuous and discrete variables.

EAs, however, are flexible enough to handle such search spaces easily. Once variation operators are known for continuous and discrete variables, constructing variation operators for mixed individuals is straightforward: Crossover, for instance, can either exchange values of corresponding variables or use the variable-level crossover operator. Many problems have been easily handled that way, like optical filter optimization, where one is looking for a number of layers, the unknown being the layer thickness (continuous) and the material the layer is made of (discrete).

### D. Artificial Creativity

But the most promising area of application of EAs, where EAs can be much more than yet another optimization method, is probably design. And here again, progress comes from the ability of EAs to handle almost any search space. The idea of component-based representations can boost innovation in structural design, architecture, and in many other areas including art. But the most original idea in that direction is that of embryogenies: The genotype is a program, and the phenotype is the result of applying that program to "grow an embryo"; the fitness is obtained by testing that phenotype in a real situation. Such an approach is already leading to astonishing results in analog circuit design for instance—though exploring a huge search space (a space of programs) implies a heavy computational cost. But we firmly believe that great achievements can come from such original ideas.

## XI. CONCLUSIONS

Natural evolution can be considered a powerful problem solver that brought *Homo sapiens* out of chaos in only a couple of billion years. Computer-based evolutionary processes can also be used as efficient problem solvers for optimization, constraint handling, machine learning, and modeling tasks. Furthermore, many real-world phenomena from the study of life, economy, and society can be investigated by simulations based on evolving systems. Last but not least, evolutionary art and design form an emerging field of applications of the Darwinian ideas. We expect that computer applications based on evolutionary principles will gain popularity in the coming years in science, business, and entertainment.

## SEE ALSO THE FOLLOWING ARTICLES

Cybernetics • Engineering, Artificial Intelligence in • Expert Systems Construction • Game Theory • Goal Programming • Hybrid Systems • Industry, Artificial Intelligence in • Intelligent Agents • Machine Learning • Neural Networks

## BIBLIOGRAPHY


Angeline, P. J., and Kinnear, K. E., Jr. (Eds.) (1996). *Advances in genetic programming II,* Cambridge, MA: The MIT Press.

Antonisse, J. (1989). A new interpretation of schema notation that overturns the binary encoding constraint. 86–91.

Bäck, Th. (1995). Generalized convergence models for tournament- and (μ, λ)-selections. *Proc. 6th International Conference on Genetic Algorithms,* L. J. Eshelman (Ed.) San Francisco: Morgan Kaufmann, 2–8.

Bäck, Th. (1996). *Evolutionary algorithms in theory and practice.* New York: Oxford University Press.

Bäck, Th., and Schütz, M. (1995). Evolution strategies for mixed-integer optimization of optical multilayer systems.

Bäck, Th., and Schwefel, H.-P. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation,* Vol. 1, No. 1, 1–23.

Bäck, Th., Rudolph, G., and Schwefel, H.-P. (1993). Evolutionary programming and evolution strategies: Similarities and differences. 11–22.

Bentley, P. J. (Ed.) (1999). *Evolutionary design by computers.* San Francisco: Morgan Kaufman.

Beyer, H. G. (1993). Toward a theory of evolution strategies: Some asymptotical results for the (1, +λ)-theory. *Evolutionary Computation,* Vol. 1, No. 2, 165–188.

Beyer, H. G. (1994). Toward a theory of evolution strategies: The (μ, λ)-theory. *Evolutionary Computation,* Vol. 2, No. 4, 381–407.

Beyer, H. G. (1995). Toward a theory of evolution strategies: On the benefit of sex—the (μ/μ, λ)-theory. *Evolutionary Computation,* Vol. 3, No. 1, 81–111.

Beyer, H. G. (1995). Toward a theory of evolution strategies: Self-adaptation. *Evolutionary Computation,* Vol. 3, No. 3, 311–347.

Blythe, P. W. (1998). Evolving robust strategies for autonomous flight: A challenge to optimal control theory. *Adaptive computing in design and manufacture,* I. Parmee (Ed.), New York: Springer Verlag, 269–283.

Bonnans, F., Gilbert, J., Lemarechal, C., and Sagastizbal, C. (1997). *Optimisation numrique, aspects thoriques et pratiques,* Vol. 23 of *Mathematiques & applications.* New York: Springer Verlag.

Cerf, R. (1996). An asymptotic theory of genetic algorithms. *Artificial evolution,* J.-M. Alliot, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers (Eds.), Vol. 1063 of *LNCS.* New York: Springer Verlag.

Chakraborty, U., Deb, K., and Chakraborty, M. (1996). Analysis of selection algorithms: A markov chain approach. *Evolutionary Computation,* Vol. 4, No. 2, 133–168.

Davidor, Y., Schwefel, H.P., and Männer, R. (Eds.) (1994). *Proceedings of the Third International Conference on Parallel Problem Solving from Nature (PPSN).* New York: Springer-Verlag.

Davis, L. (Ed.) (1987). *Genetic algorithms and simulated annealing.* San Francisco: Morgan Kaufmann.

Davis, L. (1989). Adapting operator probabilities in genetic algorithms. 61–69.

Davis, L., and Steenstrup, M. (1987). Genetic algorithms and simulated annealing: An overview. 1–11.

Davis, T. E., and Principe, J. C. (1991). A simulated annealing like convergence theory for simple genetic algorithm. *Proc. 4th International Conference on Genetic Algorithms,* R. K. Belew and L. B. Booker (Eds.). San Francisco: Morgan Kaufmann, 174–181.

Davis, T. E., and Principe, J. C. (1993). A markov chain framework for the simple genetic algorithm. *Evolutionary Computation,* Vol. 1, No. 3, 269–292.

DeJong, K. A. (1992). Are genetic algorithms function optimizers? *Proc. 2nd* Conference on Parallel Problems Solving from Nature, R. Manner and B. Manderick (Eds.). Amsterdam: North Holland, 3–13.

Eiben, A. E., Aarts, E. H. L., and Van Hee, K. M. (1991). Global convergence of genetic algorithms: A markov chain analysis. *Proc. 1st Parallel Problem Solving from Nature,* H.-P. Schwefel and R. Männer (Eds.). New York: Springer Verlag, 4–12.

Eiben, A. E., Raue, P.-E., and Ruttkay, Z. (1994). *Genetic algorithms with multi-parent recombination,* 78–87.

Eshelman, L. J. (Ed.) (1995). *Proceedings of the Sixth International Conference on Genetic Algorithms.* San Francisco: Morgan Kaufmann.

Eshelman, L. J., Caruana, R. A., and Schaffer, J. D. (1989). Biases in the crossover landscape. 10–19.

European Network on Evolutionary Computing. Successful applications of evolutionary algorithms. Available at http://evonet.dcs.napier.ac.uk/.

Fogel, D. B. (1995). *Evolutionary computation. Toward a new philosophy of machine intelligence.* Piscataway, NJ: IEEE Press.

Fogel, D. B., and Atmar, W. (1992). *Proceedings of the First Annual Conference on Evolutionary Programming,* La Jolla, CA. Evolutionary Programming Society.

Fogel, D. B., and Atmar, W. (1993). *Proceedings of the Second Annual Conference on Evolutionary Programming,* La Jolla, CA. Evolutionary Programming Society.

Fogel, D. B., and Stayton, L. C. (1994). On the effectiveness of crossover in simulated evolutionary optimization. *BioSystems,* Vol. 32, 171–182.

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

Fogel, D. B., Fogel, L. J., Atmar, W., and Fogel, G. B. (1992). Hierarchic methods of evolutionary programming. 175–182.

Galinier, P., and Hao, J. (1999). Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization,* Vol. 3, No. 4, 379–397.

Gero, J. (1998). Adaptive systems in designing: New analogies from genetics and developmental biology. *Adaptive computing in design and manufacture,* I. Parmee (Ed.). New York: Springer Verlag, 3–12.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision Sciences,* Vol. 8, No. 1, 156–166.

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

Goldberg, D. E., and Deb, K. (1991). A comparative study of selection schemes used in genetic algorithms. *Foundations of genetic algorithms,* G. J. E. Rawlins (Ed.). San Francisco: Morgan Kaufmann, 69–93.

Hamda, H., and Schoenauer, M. (2000). Adaptive techniques for evolutionary topological optimum design. *Evolutionary design and manufacture,* I. Parmee (Ed.). 123–136.

Hart, E., and Ross, P. (1998). A heuristic combination method for solving job-shop scheduling problems. *Proc. 5th Conference on Parallel Problems Solving from Nature,* T. Bäck, G. Eiben, M. Schoenauer, and H.-P. Schwefel (Eds.).

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

Jones, T. (1995). Crossover, macromutation and population-based search. 73–80.

Kinnear, K. E., Jr. (Ed.) (1994). *Advances in genetic programming,* Cambridge, MA: The MIT Press.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural evolution.* Cambridge, MA: The MIT Press.

Koza, J. R., *et al.* (1999). *Genetic programming III: Automatic synthesis of analog circuits.* Cambridge, MA: The MIT Press.

Levine, D. (1997). An evolutionary approach to airline crew scheduling. *Handbook of evolutionary computation,* T. Bäck, D. B. Fogel, and Z. Michalewicz (Eds.). New York: Oxford University Press, G9.4:1–8.

Martin, S., Rivory, J., and Schoenauer, M. (1995). Synthesis of optical multi-layer systems using genetic algorithms. *Applied Optics,* Vol. 34, 2267.

McDonnell, J. R., Reynolds, R. G., and Fogel, D. B. (Eds.) (1995). *Proceedings of the Fourth Annual Conference on Evolutionary Programming.* Cambridge, MA: The MIT Press.

Merz, P., and Freisleben, B. (1999). Fitness landscapes and memetic algorithm design. *New ideas in optimization,* D. Corne, M. Dorigo, and F. Glover (Eds.). London: McGraw-Hill, 245–260.

Michalewicz, Z. (1996). *Genetic algorithms + data structures = evolution programs,* 3rd ed. New York: Springer Verlag.

Miettinen, K., Mkel, M. M., Neittaanmki, P., and Périaux, J. (Eds.) (1999). *Evolutionary algorithms in engineering and computer science.* New York: John Wiley.

Miller, B. L., and Goldberg, D. E. (1996). Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation,* Vol. 4, No. 2, 113–132.

Mueller, S. D., *et al.* (2001). Evolution strategies for film cooling optimization. *AIAA Journal,* Vol. 39, No. 3.

Nissen, V. (1997). Quadratic assignment. *Handbook of evolutionary computation,* T. Bäck, D. B. Fogel, and Z. Michalewicz (Eds.). New York: Oxford University Press, G9.10:1–8.

Nix, A. E., and Vose, M. D. (1992). Modeling genetic algorithms with markov chains. *Annals of Mathematics and Artificial Intelligence,* Vol. 5, No. 1, 79–88.

Obayashi, S. (1997). Pareto genetic algorithm for aerodynamic design using the Navier-Stokes equations. *Genetic algorithms and evolution strategies in engineering and computer sciences,* D. Quadraglia, J. Périaux, C. Poloni, and G. Winter (Eds.). New York: John Wiley, 245–266.

Oussedik, S., and Delahaye, D. (1998). Reduction of air traffic congestion by genetic algorithms. *Proc. 5th Conference on Parallel Problems Solving from Nature,* T. Bäck, G. Eiben, M. Schoe-

nauer, and H.-P. Schwefel (Eds.). New York: Springer Verlag, 885–894.

Paechter, B., Rankin, R., Cumming, A., and Fogarty, T. C. (1998). Timetabling the classes of an entire university with an evolutionary algorithm. *Proc. 5th Conference on Parallel Problems Solving from Nature*. T. Bäck, G. Eiben, M. Schoenauer, and H.-P. Schwefel (Eds.). New York: Springer Verlag.

Parmee, I. (Ed.) (1998). *Adaptive computing in design and manufacture*. New York: Springer Verlag.

Parmee, I. (Ed.) (2000). *Adaptive computing in design and manufacture—ACDM'2000*. New York: Springer Verlag.

Périaux, J., and Winter, G. (Eds.) (1995). *Genetic algorithms in engineering and computer sciences*. New York: John Wiley.

Quadraglia, D., Périaux, J., Poloni, C., and Winter, G. (Eds.) (1997). *Genetic algorithms and evolution strategies in engineering and computer sciences*. New York: John Wiley.

Radcliffe, N. J. (1991). Equivalence class analysis of genetic algorithms. *Complex Systems,* Vol. 5, 183–220.

Rechenberg, I. (1973). *Evolutionstrategie: Optimierung technisher systeme nach prinzipien des biologischen evolution*. Stuttgart: Fromman-Holzboog Verlag.

Rosenman, M. (1999). Evolutionary case-based design. *Proc. Artificial Evolution'99,* New York: Springer-Verlag, 53–72.

Rudolph, G. (1994). Convergence analysis of canonical genetic algorithm. *IEEE Transactions on Neural Networks,* Vol. 5, No. 1, 96–101.

Rudolph, G. (1994). Convergence of non-elitist strategies. *Proc. First IEEE International Conference on Evolutionary Computation,* Z. Michalewicz, J. D. Schaffer, H.-P. Schwefel, D. B. Fogel, and H. Kitano (Eds.). New York: IEEE Press, 63–66.

Schaffer, J. D. (Ed.) (1989). *Proceedings of the Third International Conference on Genetic Algorithms*. San Francisco: Morgan Kaufmann.

Schwefel, H.-P. (1995). *Numerical optimization of computer models,* 2nd ed. New York: John Wiley & Sons.

Syswerda, G. (1991). A study of reproduction in generational and steady state genetic algorithm. *Foundations of genetic algorithms,* G. J. E. Rawlins (Ed.). San Francisco: Morgan Kaufmann, 94–101.

Törn, A., and Zilinskas, A. (1989). *Global optimization*. New York: Springer-Verlag.

Whitley, D. (1989). The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. *Proc. 3rd International Conference on Genetic Algorithms,* J. D. Schaffer (Ed.). San Francisco: Morgan Kaufmann, 116–121.

Whitley, D., Scott, V. S., and Böhm, P. W. (1997). Knapsack problems. *Handbook of evolutionary computation,* T. Bäck, D. B. Fogel, and Z. Michalewicz (Eds.). New York: Oxford University Press, G9.7:1–7.

Zalzala, A. (Ed.) (1995). *Genetic algorithms in engineering systems: Innovations and applications*. London: IEE.

Zalzala, A. (Ed.) (1995). *Second conference on genetic algorithms in engineering systems: Innovations and applications*. London: IEE.

Zitzler, E., Deb, K., Thiele, L., Corne, D., and Coello, C. (Eds.) (2001). *Proceedings of Evolutionary Multi-Critrion Optimization '01*. New York: Springer Verlag.