The Advantages of Evolutionary Computation

David B. Fogel
Natural Selection, Inc.
3333 North Torrey Pines Ct., Suite 200
La Jolla, CA 92037
dfogel@natural-selection.com

Abstract

Evolutionary computation is becoming common in the solution of difficult, real-world problems in industry, medicine, and defense. This paper reviews some of the practical advantages to using evolutionary algorithms as compared with classic methods of optimization or artificial intelligence. Specific advantages include the flexibility of the procedures, as well as the ability to self-adapt the search for optimum solutions on the fly. As desktop computers increase in speed, the application of evolutionary algorithms will become routine.

1 Introduction

Darwinian evolution is intrinsically a robust search and optimization mechanism. Evolved biota demonstrate optimized complex behavior at every level: the cell, the organ, the individual, and the population. The problems that biological species have solved are typified by chaos, chance, temporality, and nonlinear interactivities. These are also characteristics of problems that have proved to be especially intractable to classic methods of optimization. The evolutionary process can be applied to problems where heuristic solutions are not available or generally lead to unsatisfactory results. As a result, evolutionary algorithms have recently received increased interest, particularly with regard to the manner in which they may be applied for practical problem solving.

Evolutionary computation, the term now used to describe the field of investigation that concerns evolutionary algorithms, offers practical advantages to the researcher facing difficult optimization problems. These advantages are multifold, including the simplicity of the approach, its robust response to changing circumstance, its flexibility, and many other facets. This paper summarizes some of these advantages and offers suggestions in designing evolutionary algorithms for real-world problem solving. It is assumed that the reader is familiar with the basic concepts of evolutionary algorithms, and is referred to Fogel (1995), Bäck (1996), and Michalewicz (1996) for introductions.

2 Advantages of Evolutionary Computation

2.1 Conceptual Simplicity

A primary advantage of evolutionary computation is that it is conceptually simple. The main flow chart that describes every evolutionary algorithm applied for function

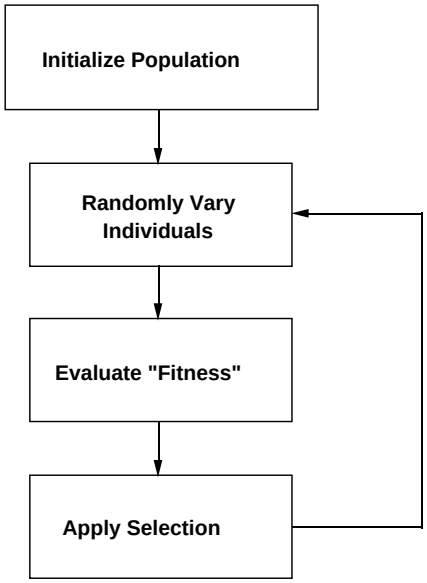

Fig. 1. The main flowchart of the vast majority of evolutionary algorithms. A population of candidate solutions to a problem at hand is initialized. This often is accomplished by randomly sampling from the space of possible solutions. New solutions are created by randomly varying existing solutions. This random variation may include mutation and/or recombination. Competing solutions are evaluated in light of a performance index describing their "fitness" (or equivalently, their error). Selection is then applied to determine which solutions will be maintained into the next generation, and with what frequency. These new "parents" are then subjected to random variation, and the process iterates.

optimization is depicted in Figure 1. The algorithm consists of initialization, which may be a purely random sampling of possible solutions, followed by iterative variation and selection in light of a performance index. This figure of merit must assign a numeric value to any possible solution such that two competing solutions can be rank ordered. Finer granularity is not required. Thus the criterion need not be specified with the precision that is required of some other methods. In particular, no gradient information needs to be presented to the algorithm. Over iterations of random variation and selection, the population can be made to converge to optimal solutions (Fogel, 1994; Rudolph, 1994; and others).

The evolutionary search is similar to the view offered by Wright (1932) involving "adaptive landscapes." A response surface describes the fitness assigned to alternative genotypes as they interact in an environment (Figure 2). Each peak corresponds to an optimized collection of behaviors (phenotypes), and thus one or more sets of optimized genotypes. Evolution probabilistically proceeds up the slopes of the topography toward peaks as selection culls inappropriate phenotypic variants. Others (Atmar, 1979; Raven and Johnson, 1986, pp. 400-401) have suggested that it is more appropriate to view the

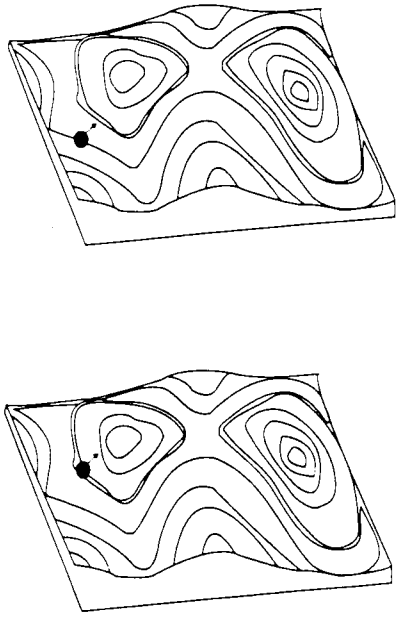

Fig. 2. Evolution on an inverted adaptive topography. A landscape is abstracted to represent the fitness of alternative phenotypes and, as a consequence, alternative genotypes. Rather than viewing the individuals or populations as maximizing fitness and thereby climbing peaks on the landscape, a more intuitive perspective may be obtained by inverting the topography. Populations proceed down the slopes of the topography toward valleys of minimal predictive error.

adaptive landscape from an inverted position. The peaks become troughs, "minimized prediction error entropy wells" (Atmar, 1979). Such a viewpoint is intuitively appealing. Searching for peaks depicts evolution as a slowly advancing, tedious, uncertain process. Moreover, there appears to be a certain fragility to an evolving phyletic line; an optimized population might be expected to quickly fall off the peak under slight perturbations. The inverted topography leaves an altogether different impression. Populations advance rapidly, falling down the walls of the error troughs until their cohesive set of interrelated behaviors is optimized. The topography is generally in flux, as a function of environmental erosion and variation, as well as other evolving organisms, and stagnation may never set in. Regardless of which perspective is taken, maximizing or minimizing, the basic evolutionary algorithm is the same: a search for the extrema of a functional describing the objective worth of alternative candidate solutions to the problem at hand.

The procedure may be written as the difference equation:

$$\mathbf{x}[t+1] = s(v(\mathbf{x}[t])) \tag{1}$$

where $\mathbf{x}[t]$ is the population at time t under a representation $\mathbf{x}$, v is a random variation

operator, and *s* is the selection operator (Fogel and Ghozeil, 1996). There are a variety of possible representations, variation operators, and selection methods (Bäck et al., 1997). Not more than 10 years ago, there was a general recommendation that the best representation was a binary coding, as this provided the greatest "implicit parallelism" (see Goldberg, 1989 for a discussion; more detail is beyond the scope of this paper). But this representation was often cumbersome to implement (consider encoding a traveling salesman problem as a string of symbols from {0,1}), and empirical results did not support any necessity, or even benefit, to binary representations (e.g., Davis, 1991; Michalewicz, 1992). Moreover, the suggestions that recombining alternative solutions through crossover operators, and amplifying solutions based on the relative fitness also did not obtain empirical support (e.g., Fogel and Atmar, 1990; Bäck and Schwefel, 1993; Fogel and Stayton, 1994; and many others). Recent mathematical results have proved that there can be no best choice for these facets of an evolutionary algorithm that would hold across all problems (Wolpert and Macready, 1997), and even that there is no best choice of representation for any individual problem (Fogel and Ghozeil, 1997). The effectiveness of an evolutionary algorithm depends on the interplay between the operators *s* and *v* as applied to a chosen representation **x** and initialization **x**[0]. This dependence provides freedom to the human operator to tailor the evolutionary approach for their particular problem of interest.

2.2 Broad Applicability

Evolutionary algorithms can be applied to virtually any problem that can be formulated as a function optimization task. It requires a data structure to represent solutions, a performance index to evaluate solutions, and variation operators to generate new solutions from old solutions (selection is also required but is less dependent on human preferences). The state space of possible solutions can be disjoint and can encompass infeasible regions, and the performance index can be time varying, or even a function of competing solutions extant in the population. The human designer can choose a representation that follows their intuition. In this sense, the procedure is representation independent, in contrast with other numerical techniques which might be applicable for only continuous values or other constrained sets. Representation should allow for variation operators that maintain a behavioral link between parent and offspring. Small changes in the structure of a parent should lead to small changes in the resulting offspring, and likewise large changes should engender gross alterations. A continuum of possible changes should be allowed such that the effective "step size" of the algorithm can be tuned, perhaps online in a self-adaptive manner (discussed later). This flexibility allows for applying essentially the same procedure to discrete combinatorial problems, continuous-valued parameter optimization problems, mixed-integer problems, and so forth.

2.3 Outperform Classic Methods on Real Problems

Real-world function optimization problems often (1) impose nonlinear constraints, (2) require payoff functions that are not concerned with least squared error, (3) involve nonstationary conditions, (4) incorporate noisy observations or random processing, or

include other vagaries that do not conform well to the prerequisites of classic optimization techniques. The response surfaces posed in real-world problems are often multimodal, and gradient-based methods rapidly converge to local optima (or perhaps saddle points) which may yield insufficient performance. For simpler problems, where the response surface is, say, strongly convex, evolutionary algorithms do not perform as well as traditional optimization methods (Bäck, 1996). But this is to be expected as these techniques were designed to take advantage of the convex property of such surfaces. Schwefel (1995) has shown in a series of empirical comparisons that in the obverse condition of applying classical methods to multi-modal functions, evolutionary algorithms offer a significant advantage. In addition, in the often encountered case of applying linear programming to problems with nonlinear constraints, this offers an almost certainly incorrect result because the assumptions required for the technique are violated. In contrast, evolutionary computation can directly incorporate arbitrary constraints (Michalewicz, 1996).

Moreover, the problem of defining the payoff function for optimization lies at the heart of success or failure: Inappropriate descriptions of the performance index lead to generating the right answer for the wrong problem. Within classic statistical methods, concern is often devoted to minimizing the squared error between forecast and actual data. But in practice, equally correct predictions are not of equal worth, and errors of identical magnitude are not equally costly. Consider the case of correctly predicting that a particular customer will ask to purchase 10 units of a particular product (e.g., an aircraft engine). This is typically worth less than correctly predicting that the customer will seek to purchase 100 units of that product, yet both predictions engender zero error and are weighted equally in classic statistics. Further, the error of predicting the customer will demand 10 units and having them actually demand 100 units is not of equal cost to the manufacturer as predicting the customer will demand 100 units and having them demand 10. One error leaves a missed opportunity cost while the other leaves 90 units in a warehouse. Yet again, under a squared error criterion, these two situations are treated identically. In contrast, within evolutionary algorithms, any definable payoff function can be used to judge the appropriateness of alternative behaviors. There is no restriction that the criteria be differentiable, smooth, or continuous.

2.4 Potential to Use Knowledge and Hybridize with other Methods

It is always reasonable to incorporate domain-specific knowledge into an algorithm when addressing particular real-world problems. Specialized algorithms can outperform unspecialized algorithms on a restricted domain of interest (Wolpert and Macready, 1997). Evolutionary algorithms offer a framework such that it is comparably easy to incorporate such knowledge. For example, specific variation operators may be known to be useful when applied to particular representations (e.g., 2-OPT on the traveling salesman problem). These can be directly applied as mutation or recombination operations. Knowledge can also be implemented into the performance index, in the form of known physical or chemical properties (e.g., van der Waals interactions, Gehlhaar et al., 1995). Incorporating such information focuses the evolutionary search, yielding a more efficient exploration of the state space of possible solutions.

Evolutionary algorithms can also be combined with more traditional optimization techniques. This may be as simple as the use of a conjugate-gradient minimization used after primary search with an evolutionary algorithm (e.g., Gehlhaar et al., 1995), or it may involve simultaneous application of algorithms (e.g., the use of evolutionary search for the structure of a model coupled with gradient search for parameter values, Harp et al., 1989). There may also be a benefit to seeding an initial population with solutions derived from other procedures (e.g., a greedy algorithm, Fogel and Fogel, 1996). Further, evolutionary computation can be used to optimize the performance of neural networks (Angeline et al., 1994), fuzzy systems (Haffner and Sebald, 1993), production systems (Wilson, 1995), and other program structures (Koza, 1992; Angeline and Fogel, 1997). In many cases, the limitations of conventional approaches (e.g., the requirement for differentiable hidden nodes when using back propagation to train a neural network) can be avoided.

2.5 Parallelism

Evolution is a highly parallel process. As distributed processing computers become more readily available, there will be a corresponding increased potential for applying evolutionary algorithms to more complex problems. It is often the case that individual solutions can be evaluated independently of the evaluations assigned to competing solutions. The evaluation of each solution can be handled in parallel and only selection (which requires at least pairwise competition) requires some serial processing. In effect, the running time required for an application may be inversely proportional to the number of processors. Regardless of these future advantages, current desktop computing machines provide sufficient computational speed to generate solutions to difficult problems in reasonable time (e.g., the evolution of a neural network for classifying features of breast carcinoma involving over 5 million separate function evaluations requires only about three hours on a 200 MHz 604e PowerPC, Fogel et al., 1997).

2.6 Robust to Dynamic Changes

Traditional methods of optimization are not robust to dynamic changes in the environment and often require a complete restart in order to provide a solution (e.g., dynamic programming). In contrast, evolutionary algorithms can be used to adapt solutions to changing circumstance. The available population of evolved solutions provides a basis for further improvement and in most cases it is not necessary, nor desirable, to reinitialize the population at random. Indeed, this procedure of adapting in the face of a dynamic environment can be used to advantage. For example, Wieland (1990) used a genetic algorithm to evolve recurrent neural networks to control a cart-pole system comprising two poles (Figure 3). The degree of difficulty depended on the relative pole lengths (i.e., the closer the poles were to each other in length, the more difficult the control problem). Wieland (1990) developed controllers for a case of one pole of length 1.0 m and the other of 0.9 m by successively controlling systems where the shorter pole was started at 0.1 m and incremented sequentially to 0.9 m. At each increment, the evolved population of networks served as the basis for a new set of controllers. A simi-

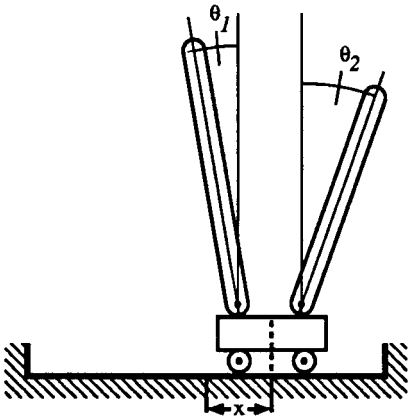

Fig. 3. A cart with two poles (Wieland, 1990). The objective is to maintain the cart between the limits of the track while not allowing either pole to exceed a specified maximum angle of deflection. The only control available is a force with which to push or pull on the cart. The difficulty of the problem is dependent on the similarity in pole lengths. Wieland (1990) and Saravanan and Fogel (1994) used evolutionary algorithms to optimize neural networks to control this plant for pole lengths of 1.0 m and 0.9 m. The evolutionary procedure required starting with poles of 1.0 m and 0.1 m, and iteratively incrementing the length of the shorter pole in a series of dynamic environments. In each case, the most recent evolved population served as the basis for new trials, even when the pole length was altered.

lar procedure was offered in Saravanan and Fogel (1994), and Fogel (1996).

The ability to adapt on the fly to changing circumstance is of critical importance to practical problem solving. For example, suppose that a particular simulation provides perfect fidelity to an industrial production setting. All workstations and processes are modeled exactly, and an algorithm is used to find a "perfect" schedule to maximize production. This perfect schedule will, however, never be implemented in practice because by the time it is brought forward for consideration, the plant will have changed: machines may have broken down, personnel may not have reported to work or failed to keep adequate records of prior work in progress, other obligations may require redirecting the utilization of equipment, and so forth. The "perfect" plan is obsolete before it is ever implemented. Rather than spend considerable computational effort to find such perfect plans, a better prescription is to spend less computational effort to discover suitable plans that are robust to expected anomalies and can be evolved on the fly when unexpected events occur.

2.7 Capability for Self-Optimization

Most classic optimization techniques require appropriate settings of exogenous variables. This is true of evolutionary algorithms as well. However, there is a long history of using the evolutionary process itself to optimize these parameters as part of the search for optimal solutions (Reed et al., 1967; Rosenberg, 1967; and others). For ex-

ample, suppose a search problem requires finding the real-valued vector that minimizes a particular functional $F(\mathbf{x})$, where $\mathbf{x}$ is a vector in n dimensions. A typical evolutionary algorithm (Fogel, 1995) would use Gaussian random variation on current parent solutions to generate offspring:

$$x'_i = x_i + \sigma_i N(0,1)$$

where the subscript indicates the ith dimension, and σ_i is the standard deviation of a Gaussian random variable. Setting the "step size" of the search in each dimension is critical to the success of the procedure (Figure 4). This can be accomplished in the following two-step fashion:

$$\sigma'_i = \sigma_i exp(\tau N(0,1)+\tau' N_i(0,1))$$
$$x'_i = x_i + \sigma'_i N_i(0,1)$$

where $\tau \propto (2n)^{0.5}$ and $\tau' \propto (2n^{0.5})^{0.5}$ (Bäck and Schwefel, 1993). In this manner, the standard deviations are subject to variation and selection at a second level (i.e., the level of how well they guide the search for optima of the functional $F(\mathbf{x})$). This general procedure has also been found effective in addressing discrete optimization problems (Angeline et al., 1996; Chellapilla and Fogel, 1997; and others). Essentially, the effect is much like a temperature schedule in simulated annealing, however, the schedule is set by the algorithm as it explores the state space, rather than *a priori* by a human operator.

2.8 Able to Solve Problems that have no known Solutions

Perhaps the greatest advantage of evolutionary algorithms comes from the ability to address problems for which there are no human experts. Although human expertise should be used when it is available, it often proves less than adequate for automating problem-solving routines. Troubles with such expert systems are well known: the experts may not agree, may not be self-consistent, may not be qualified, or may simply be in error. Research in artificial intelligence has fragmented into a collection of methods and tricks for solving particular problems in restricted domains of interest. Certainly, these methods have been successfully applied to specific problems (e.g., the chess program Deep Blue). But most of these applications require human expertise. They may be impressively applied to difficult problems requiring great computational speed, but they generally do not advance our understanding of intelligence. "They solve problems, but they do not solve the problem of how to solve problems," (Fogel, 1995, p. 259). In contrast, evolution provides a method for solving the problem of how to solve problems. It is a recapitulation of the scientific method (Fogel et al., 1966) that can be used to learn fundamental aspects of any measurable environment.

3 Conclusions

Although the history of evolutionary computation dates back to the 1950s and 1960s (Fogel, 1995, ch. 3), only within the last decade have evolutionary algorithms become

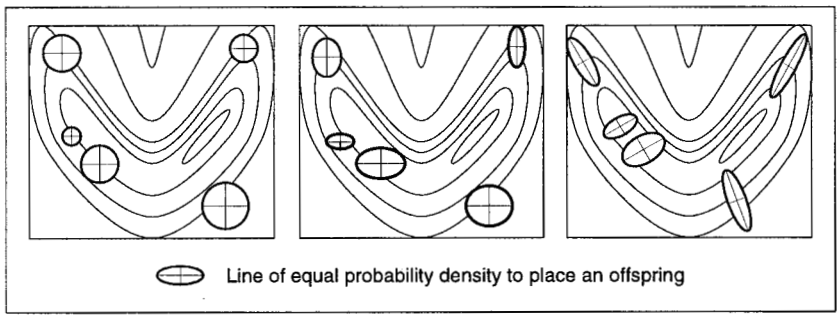

Fig. 4. When using Gaussian mutations in all dimensions (as in evolution strategies or evolutionary programming), the contours of equal probability density for placing offspring are depicted above (Bäck, 1996). In the left panel, all standard deviations in each dimension are equal, resulting in circular contours. In the middle panel, the standard deviations in each dimenion may vary, but the perturbation in each dimension is independent of the others (zero covariance), resulting in elliptical contours. In the right panel, arbitrary covariance is applied, resulting in contours that are rotatable ellipses. The method of self-adaptation described in text can be extended to adapt arbitrary covariances, thereby allowing the evolutionary algorithm to adapt to the changes in the response surface during the search for the optimum position on the surface. Similar procedures have been offered for self-adaptation when solving discrete combinatorial problems.

practicable for solving real-world problems on desktop computers (Bäck et al., 1997). As computers continue to deliver accelerated performance, these applications will only become more routine. The flexibility of evolutionary algorithms to address general optimization problems using virtually any reasonable representation and performance index, with variation operators that can be tailored for the problem at hand, and selection mechanisms tuned for the appropriate level of stringency, gives these techniques an advantage over classic numerical optimization procedures. Moreover, the two-step procedure to self-adapt parameters that control the evolutionary search frees the human operator from having to handcraft solutions, which would often be time consuming or simply infeasible. Evolutionary algorithms offer a set of procedures that may be usefully applied to problems that have resisted solution by common techniques, and can be hybridized with such techniques when such combinations appear beneficial.

Acknowledgments

The author would like to thank the conference organizers for inviting this presentation and P.J. Angeline for his comments and suggestions.

References

1. Angeline, P.J., Fogel, D.B.: An evolutionary program for the identification of dynamical systems. *SPIE Aerosence 97, Symp. on Neural Networks*, S.K. Rogers and D. Ruck (eds.), Vol. 3077 (1997) 409-417.
2. Angeline, P.J., Fogel, D.B., Fogel, L.J.: A comparison of self-adaptation methods for finite state machines in a dynamic environment. *Evolutionary Programming V*, L.J. Fogel, P.J. Angeline, T. Bäck (eds.), MIT Press, Cambridge, MA (1996) 441-449.
3. Angeline, P.J., Saunders, G.M., Pollack, J.B.: An evolutionary algorithm that constructs recurrent neural networks. *IEEE Trans. Neural Networks*, **5** (1994) 54-65.
4. Atmar, W.: The inevitability of evolutionary invention. (1979) unpublished manuscript.
5. Bäck, T.: *Evolutionary Algorithms in Theory and Practice*, Oxford, NY (1996).
6. Bäck, T., Fogel, D.B., Michalewicz, Z. (eds.): *Handbook of Evolutionary Computation*, Oxford, NY (1997).
7. Bäck, T., Schwefel, H.-P.: An overview of evolutionary algorithms for parameter optimization. *Evol. Comp.* **1** (1993) 1-24.
8. Chellapilla, K., Fogel, D.B.: Exploring self-adaptive methods to improve the efficiency of generating approximate solutions to traveling salesman problems using evolutionary programming. *Evolutionary Programming VI*, P.J. Angeline, R.G. Reynolds, J.R. McDonnell, and R. Eberhart (eds.), Springer, Berlin (1997) 361-371.
9. Davis, L. (ed.): *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, NY (1991)
10. Fogel, D.B.: Asymptotic convergence properties of genetic algorithms and evolutionary programming: analysis and experiments. *Cybern. & Syst.*, **25** (1994) 389-407.
11. Fogel, D.B.:*Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE Press, NY (1995).
12. Fogel, D.B.: A 'correction' to some cart-pole experiments. *Evolutionary Programming VI*, L.J. Fogel, P.J. Angeline, and T. Bäck (eds.), MIT Press, Cambridge, MA (1996) 67-71.
13. Fogel, D.B., Atmar, J.W.: Comparing genetic operators with Gaussian mutations in simulated evolutionary processes using linear systems. *Biol. Cybern.*, **63** (1990) 111-114.
14. Fogel, D.B., Fogel, L.J.: Using evolutionary programming to schedule tasks on a suite of heterogeneous computers. *Comp. & Oper. Res.*, **23** (1996) 527-534.
15. Fogel, D.B., Ghozeil, A.: Using fitness distributions to design more efficient evolutionary computations. *Proc. of 1996 IEEE Conf. on Evol. Comp.*, Keynote Lecture, IEEE Press, NY (1996) 11-19.
16. Fogel, D.B., Ghozeil, A.: A note on representations and variation operators," *IEEE Trans. Evol. Comp.*, **1** (1997) in press.
17. Fogel, D.B., Stayton, L.C.: On the effectiveness of crossover in simulated evolutionary optimization. *BioSystems*, **32** (1994) 171-182.

18. Fogel, D.B., Wasson, E.C., Boughton, E.M., and Porto, V.W.: A step toward computer-assisted mammography using evolutionary programming and neural networks. *Cancer Lett.* (1997) in press.
19. Fogel, L.J., Owens, A.J., Walsh, M.J.: *Artificial Intelligence through Simulated Evolution*, John Wiley, NY (1996).
20. Gehlhaar, D.K., Verkhivker, G.M., Rejto, P.A., Sherman, C.J., Fogel, D.B., Fogel, L.J., Freer, S.T.: Molecular recognition of the inhibitor AG-1343 by HIV-1 protease: conformationally flexible docking by evolutionary programming. *Chem. & Biol.*, **2** (1995) 317-324.
21. Haffner, S.B., Sebald, A.V.: Computer-aided design of fuzzy HVAC controllers using evolutionary programming. *Proc. of the 2nd Ann. Conf. on Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA (1993) 98-107.
22. Harp, S.A., Samad, T., Guha, A.: Towards the genetic synthesis of neural networks. *Proc. of the 3rd Intern. Conf. on Genetic Algorithms*, J.D. Schaffer (ed.), Morgan Kaufmann, San Mateo, CA (1989) 360-369.
23. Koza, J.R.:*Genetic Programming*, MIT Press, Cambridge, MA (1992).
24. Michalewicz, Z.:*Genetic Algorithms + Data Structures = Evolution Programs*, Springer, Berlin (1992).
25. Michalewicz, Z.:*Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed., Springer, Berlin (1996).
26. Raven, P.H., Johnson, G.B.: *Biology*, Times Mirror, St. Louis, MO (1986).
27. Reed, J., Toombs, R., Barricelli, N.A.: Simulation of biological evolution and machine learning. *J. Theor. Biol.*, 17 (1967) 319-342.
28. Rosenberg, R.: Simulation of genetic populations with biochemical properties," Ph.D. Dissertation, Univ. of Michigan, Ann Arbor (1967).
29. Rudolph, G.: Convergence analysis of canonical genetic algorithms. *IEEE Trans. Neural Networks*, **5** (1994) 96-101.
30. Saravanan, N., Fogel, D.B.: Evolving neurocontrollers using evolutionary programming. *IEEE Conf. on Evol. Comp.*, Vol. 1, IEEE Press, Piscataway, NJ (1994) 217-222.
31. Schwefel, H.-P.: *Evolution and Optimum Seeking*, John Wiley, NY (1995).
32. Wieland, A.P.: Evolving controls for unstable systems. *Connectionist Models: Proceedings of the 1990 Summer School*, D.S. Touretzky, J.L. Elman, T.J. Sejnowski, and G.E. Hinton(eds.), Morgan Kaufmann, San Mateo, CA (1990) 91-102.
33. Wilson, S.W.: Classifier fitness based on accuracy. *Evol. Comp.*, **3** (1995) 149-175.
34. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. *IEEE Trans. Evol. Comp.*, **1** (1997) in press.
35. Wright, S.: The roles of mutation, inbreeding, crossbreeding, and selection in evolution. *Proc. 6th Int. Cong. Genetics*, Vol. 1, Ithaca (1932) 356-366.