Annu. Rev. Ecol. Syst. 1999. 30:593–616

EVOLUTIONARY COMPUTATION: An Overview

Melanie Mitchell* and Charles E. Taylor**
*Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, New Mexico 87501; e-mail: mm@santafe.edu, **Department of Organismic Biology, Ecology, and Evolution, University of California, Los Angeles, California 90095; e-mail: taylor@biology.ucla.edu

Key Words artificial life, computational modeling

INTRODUCTION

Evolutionary computation is an area of computer science that uses ideas from biological evolution to solve computational problems. Many such problems require searching through a huge space of possibilities for solutions, such as among a vast number of possible hardware circuit layouts for a configuration that produces desired behavior, for a set of equations that will predict the ups and downs of a financial market, or for a collection of rules that will control a robot as it navigates its environment. Such computational problems often require a system to be adaptive—that is, to continue to perform well in a changing environment.

Problems like these require complex solutions that are usually difficult for human programmers to devise. Artificial intelligence practitioners once believed that it would be straightforward to encode the rules that would confer intelligence on a program; expert systems were one result of this early optimism. Nowadays, however, many researchers believe that the "rules" underlying intelligence are too complex for scientists to encode by hand in a top-down fashion. Instead they believe that the best route to artificial intelligence and other difficult computational problems is through a bottom-up paradigm in which humans write only very simple rules and provide a means for the system to adapt. Complex behaviors such as intelligence will emerge from the parallel application and interaction of these rules. Neural networks are one example of this philosophy; evolutionary computation is another.

Biological evolution is an appealing source of inspiration for addressing difficult computational problems. Evolution is, in effect, a method of searching among an enormous number of possibilities—e.g., the set of possible gene sequences—for "solutions" that allow organisms to survive and reproduce in their environments. Evolution can also be seen as a method for adapting to changing environments. And, viewed from a high level, the "rules" of evolution are remarkably simple: Species evolve by means of random variation (via mutation, recombination, and

0066-4162/99/1120-0593$08.00

other operators), followed by natural selection in which the fittest tend to survive and reproduce, thus propagating their genetic material to future generations. Yet these simple rules are thought to be responsible for the extraordinary variety and complexity we see in the biosphere.

There are several approaches that have been followed in the field of evolutionary computation. The general term for such approaches is *evolutionary algorithms*. The most widely used form of evolutionary algorithms are *genetic algorithms* (GAs), which will be the main focus of this review. Other common forms of evolutionary algorithms will be described in the third section.

A SIMPLE GENETIC ALGORITHM

The simplest version of a genetic algorithm consists of the following components:

1. A population of candidate solutions to a given problem (e.g., candidate circuit layouts), each encoded according to a chosen representation scheme (e.g., a bit string encoding a spatial ordering of circuit components). The encoded candidate solutions in the population are referred to metaphorically as chromosomes, and units of the encoding (e.g., bits) are referred to as genes. The candidate solutions are typically haploid rather than diploid.
2. A fitness function that assigns a numerical value to each chromosome in the population measuring its quality as a candidate solution to the problem at hand.
3. A set of genetic operators to be applied to the chromosomes to create a new population. These typically include selection, in which the fittest chromosomes are chosen to produce offspring; crossover, in which two parent chromosomes recombine their genes to produce one or more offspring chromosomes; and mutation, in which one or more genes in an offspring are modified in some random fashion.

A typical GA carries out the following steps:

1. Start with a randomly generated population of n chromosomes.
2. Calculate the fitness $f(x)$ of each chromosome x in the population.
3. Repeat the following steps until n offspring have been created:
   (*a*) Select a pair of parent chromosomes from the current population, the probability of selection increasing as a function of fitness.
   (*b*) With probability p_c (the crossover probability), cross over the pair by taking part of the chromosome from one parent and the other part from the other parent. This forms a single offspring.
   (*c*) Mutate the resulting offspring at each locus with probability p_m (the mutation probability) and place the resulting chromosome in the new population. Mutation typically replaces the current value of a locus (e.g., 0) with another value (e.g., 1).

4. Replace the current population with the new population.
5. Go to step 2.

Each iteration of this process is called a generation. A genetic algorithm is typically iterated for anywhere from 50 to 500 or more generations. The entire set of generations is called a run. At the end of a run, there are typically one or more highly fit chromosomes in the population. Since randomness plays a large role in each run, two runs with different random-number seeds will generally produce different detailed behaviors.

The simple procedure just described is the basis for most applications of GAs. There are a number of details to fill in, such as how the candidate solutions are encoded, the size of the population, the details and probabilities of the selection, crossover, and mutation operators, and the maximum number of generations allowed. The success of the algorithm depends greatly on these details.

A BRIEF HISTORY OF EVOLUTIONARY COMPUTATION

In the 1950s and the 1960s several computer scientists independently invented different evolutionary computation methods. In the 1960s, Rechenberg introduced evolution strategies (61), a method he used to optimize real-valued parameters for devices such as airfoils. A "population" consisted of two individuals—a parent and a child mutated from the parent—each encoding a set of real-valued parameters to be optimized. The fitter of the two was selected to be the parent for the next generation. Mutation consisted of incrementing or decrementing a real value according to a given distribution. The parameters of this distribution were themselves encoded as part of each individual and thus coevolved with the parameters to be optimized. This idea was further developed by Schwefel (65, 66), and the theory and application of evolution strategies has remained an active area of research (see, e.g., 6).

Fogel, Owens, & Walsh developed *evolutionary programming* (23), a technique in which candidate solutions to given tasks were represented as finite-state machines and were evolved by randomly mutating their state-transition diagrams and selecting the fittest. As in evolution strategies, random mutation was the only source of variation. A somewhat broader formulation of evolutionary programming also remains an area of active research (see, e.g., 21).

The techniques called genetic algorithms (GAs) were first invented by Holland in the 1960s (31). GAs are population-based algorithms in which mutation and crossover are sources of random variation. GAs typically worked on strings of bits rather than real-valued parameters. The simple GA given above is close to the algorithm proposed by Holland. Holland's original proposal also included an "inversion" operator for reordering of bits on a chromosome. In contrast with evolution strategies and evolutionary programming, Holland's goal was not to design algorithms to solve specific problems, but rather to formally study the

phenomenon of adaptation as it occurs in nature and to develop ways in which the mechanisms of natural adaptation might be imported into computer systems. Several other people working in the 1950s and 1960s developed evolution-inspired algorithms for optimization and machine learning; see (22, 26, 49) for discussions of this history.

In the last several years there has been widespread interaction among researchers studying various evolutionary computation methods, and the boundaries between evolution strategies, evolutionary programming, genetic algorithms, and other evolutionary computation methods have broken down to some extent. In this review most of our examples involve what researchers have called genetic algorithms, though in many cases these will be of a somewhat different form than Holland's original proposal.

In this review we first survey some applications of evolutionary computation in business, science, and education, and we conclude with a discussion of evolutionary computation research most relevant to problems in evolutionary biology. Due to space limitations, we do not survey the extensive work that has been done on the theoretical foundations of evolutionary computation; much work in this area can be found in the various *Foundations of Genetic Algorithms* proceedings volumes (11, 58, 76, 77) and in recent reviews (6, 21, 26, 49).

COMMERCIAL APPLICATIONS OF EVOLUTIONARY ALGORITHMS

We suggested in the introduction that evolution can be viewed as a method for searching through enormous numbers of possibilities in parallel, in order to find better solutions to computational problems. It is a way to find solutions that, if not necessarily optimal, are still good—what Simon (68) has termed "satisficing." Problems where it is sufficient to have satisficing solutions arise frequently in business and engineering. For many of these applications the traditional methods of search (e.g., hill-climbing and conjugate-gradient methods) perform well (55). It is well established that no single method of optimization is best for all applications—the best method will depend on the problem and computational devices at hand. We are aware of no good theory that will generally predict when one method or another is best for any particular problem. It has been speculated that evolutionary algorithms perform relatively well when there is a large number of parameters to be determined and when there is a high degree of epistasis so the adaptive surface of solutions is complex, having many intermediate optima. Lewontin (44) has stated that evolutionary computation has not solved any problems that could not be solved by traditional means. This might be true or not—we know of no tests of this statement. Nevertheless, evolutionary computation is finding use in a variety of commercial and scientific applications, and that number is certainly growing. Some examples include integrated circuit design, factory

scheduling, robot sensor design, image processing, financial market prediction, and drug design.

For example, a common problem in drug design is to learn how well an arbitrary ligand will bind to some given enzyme. This problem is known to be NP-complete (52). A particular problem of great significance is to know how well inhibitors will bind to HIV protease enzymes. When an HIV-1 virus matures, it must cleave proteins manufactured by the viral genetic material into the individual proteins required by HIV-1. Such cleavage takes place at the active site in the protease. Protease inhibitors bind tightly into this active site and disrupt the life cycle of the virus. Such inhibitors are finding widespread use in treating AIDS. The market for protease inhibitors is thus huge. Companies would like to screen candidate molecules and determine whether they will fit into the active site and how well they will bind there. Natural Selection Inc. provided software to Aguron Pharmaceuticals that combines models of ligand-protein interactions for molecular recognition with evolutionary programming to search the space of all possible configurations of the ligand-protein complex (24). Each candidate solution of the evolving population is a vector with rigid-body coordinates and the angles about its rotatable bonds. Those individuals with the lowest calculated energies are then used as parents for the next generation. Mutations occur as random changes in the bond angles in the offspring candidates. In practice it is useful to have the magnitudes of the mutations themselves evolve as well.

Supply-chain management provides examples of a very different sort. For instance, Volvo trucks are built to order and have dozens of options for each tractor cab, giving millions of configurations that must be scheduled and inventory checked, after which tools and time must be provided at the appropriate place on the plant floor and then delivered there. Starting from a collection of average-quality schedules, the scheduling program provided by I2 Technologies evolves a satisficing schedule for plant production each week (57). Deere & Company was probably the first to use such methods (54), also provided by I2 Technologies, for making their John Deere tractors, and now employs the methods in six of their assembly plants.

Evolution in such cases is based on an optimizing-scheduling procedure that was developed and employed at the US Navy's Point Magu Naval Airbase. Each chromosome in the evolving population encodes a schedule—a permutation of the tasks to be performed—whose fitness is the cost of the schedule after it is further optimized by an additional scheduling program.

An altogether different problem is to predict stock market prices. Several large corporations, including Citibank, Midland Bank, and Swiss Bank (through their partner Prediction Company), have been evolving programs that attempt such predictions (38). Typically such methods involve backcasting—withdrawing the most recent data from the evaluators, then determining how well each program in the population predicts that data. Not surprisingly, the details of how such programs work, including their performance, are trade secrets, though, for

reasons discussed in the section on the Baldwin effect, it seems that such programs are likely to contain some other sorts of learning mechanisms in addition to evolution.

Asahi Microsystems is building an evolvable hardware (EHW) chip that is part of an integrated circuit for cellular telephones. When computer chips are made, there is slight variation from chip to chip, due to differences in capacitance, resistance, etc. A percentage of such chips do not perform up to specification and so must be discarded. In general, as chips are made smaller they are less likely to perform to specification. In the case of analog cellular telephones, where size is a major issue, certain filters must perform to within 1% of the central frequencies. The laboratory of T Higuchi at Tsukuba (50) has shown how to build chips that are tunable with 38 parameters as they come off the assembly line. Their EHW microchip will alter these parameters using evolutionary computation and then set the appropriate switches on a field programmable gate array (FPGA). The EHW chip is leading to improved yield rates of the filter chip and also will lead to smaller circuits and power consumption. The chip is to appear in January 1999 with a target production of 400,000 chips per month.

Descriptions of various other commercial applications can be found in journals such as *Evolutionary Computation* or *IEEE Transactions on Evolutionary Computing*, and in various conference proceedings (e.g., 3, 7).

SCIENTIFIC APPLICATIONS OF EVOLUTIONARY ALGORITHMS

In addition to their commercial applications, evolutionary algorithms have been used extensively in science, both as search methods for solving scientific problems and as scientific models of evolutionary systems including population genetics, clonal selection in immune systems, innovation and the evolution of strategies in economic systems, and the evolution of collective behavior in social systems. In this section we review two scientific applications, one in ecology and one in computer science.

Using Genetic Programming to Evolve Optimal Foraging Strategies

Koza, Rice, & Roughgarden used a type of evolutionary algorithm (called genetic programming) to investigate two ecological questions: What makes for an optimal foraging strategy, and how can an evolutionary process assemble strategies that require complex calculations from simple components? (41). They addressed this by building on Roughgarden's work on foraging strategies of Anolis lizards. These lizards wait in trees for insects to appear in their field of vision and then pursue and eat those insects they deem most desirable.

The model strategies involved four variables: the abundance a of insects (the probability of an insect appearing per square meter per second, assumed to be uniform over space and time), the sprint velocity v of the lizard (assumed to be constant), and the coordinates x, y of the insect in the lizard's view, assumed in this case to be two dimensional. (It is also assumed that only one insect at a time is viewed, all insects are identical, and that each chase consists of the lizard leaving and returning to its perch before a new insect appears). A *strategy* is a function of these four variables that returns 1 if the insect is to be chased, -1 otherwise. The goal is to devise a function that maximizes food capture per unit time. Clearly not every insect is worth chasing; if an insect is too small or too far away it will take too much time to catch it and might even be gone by the time the lizard reaches it.

In Koza, Rice, & Roughgarden's model, a single simulation of a lizard's behavior consisted of assigning values to the variables a and v, and then allowing 300 simulated seconds in which insects appear at different x, y coordinates (with uniform probability over x and y); the lizard uses its strategy to decide which ones to chase. In one of Koza, Rice, & Roughgarden's experiments, the lizard's 10×20 meter viewing area was divided into three regions, as shown in Figure 1(a). Insects appearing in region I always escaped when chased; those appearing in region II never escaped; and those appearing in region III escaped with probability zero on the x axis and linearly increasing with angle to a maximum of 0.5 on the y axis. The optimal strategy is for a lizard to always ignore insects in region 1, chase those in region 2 that are sufficiently close, and chase those in region 3 that are

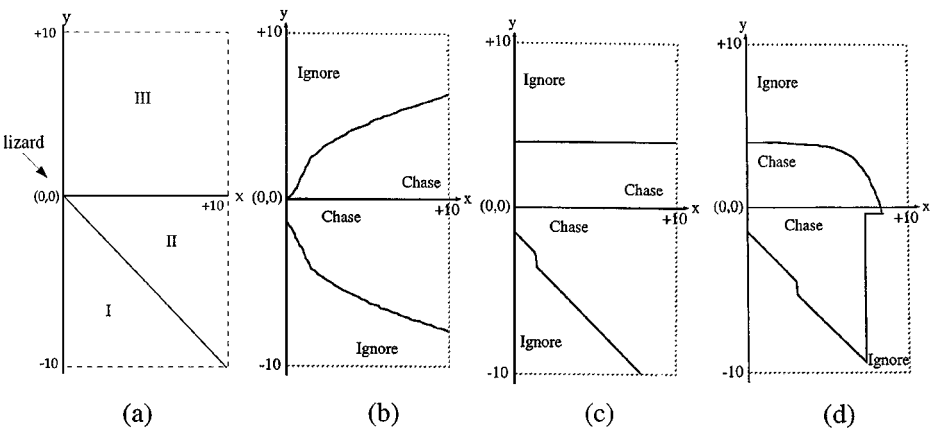

Figure 1 (a) The lizard's viewing area, divided into three regions, each with a different escape probability for insects. (b)–(d) Switching curves to illustrate the behavior of the best program in the population at generations 0 (b), 12 (c), and 46 (d) from one run of the genetic programming algorithm. The curves divide the lizard's viewing area into regions in which it will chase insects and regions in which it will ignore insects. (Adapted from 41.)

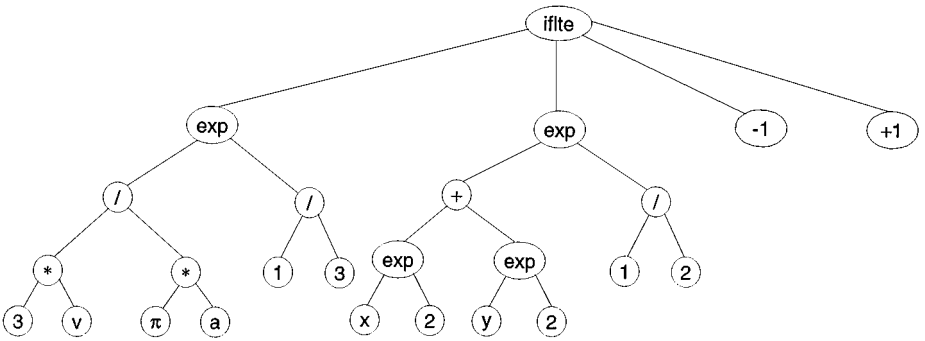

Figure 2 The optimal strategy for a simpler version of the foraging problem, encoded as a parse tree.

sufficiently close and have a high-enough probability of not escaping. No analytic form for the optimal strategy in this model was known.

The genetic programming algorithm is a type of genetic algorithm devised by Koza (40) in which the individuals undergoing evolution are computer programs encoded as parse trees. For example, Koza, Rice, & Roughgarden derived by hand the following formula for an optimal strategy in a simpler version of the problem in which insects never escape in any region:

$$Sig\left[\left(\frac{3v}{\pi a}\right)^{\frac{1}{3}} - (x^2 + y^2)^{\frac{1}{2}}\right], \qquad 1.$$

where $Sig(z)$ returns $+1$ if z is positive and -1 otherwise. Figure 2 gives this same strategy in the form of a parse tree (similar to those formed by compilers when parsing computer programs), in which nodes contain either functions (e.g., $+$, $-$, exponentiation) or the terminals on which the functions are to be performed (e.g., v, a, π, 1, 2, 3). In Figure 2, *exp* is an exponentiation operator that takes the absolute value of its first argument and raises that to its second argument, and *iflte* is the "if less than or equal to" operator: If its first argument is less than or equal to its second argument, it returns its third argument; otherwise it returns its fourth argument.

The objective is to discover a structure like that in Figure 2 that encodes an optimal (or at least good) strategy for the foraging task. It starts by forming a population of M randomly generated programs in parse-tree format. The fitness of each population member is then calculated by simulating the behavior of the corresponding strategy for several values of the variables (fitness cases). The highest fitness individuals then reproduce—either by direct copying or by crossover (no mutation was used)—to produce a new population; this whole evolutionary process is iterated for many generations.

The initial random programs are generated using a predefined set of functions and terminals. One of the arts of genetic programming is figuring out what should

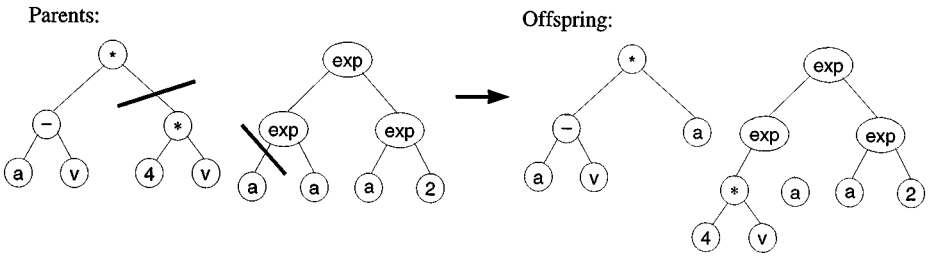

Figure 3 Illustration of crossover between two parse trees to produce two offspring. The (randomly chosen) point of crossover on each tree is marked by a dark line.

be in this set. Koza, Rice, & Roughgarden's set of terminals was $\{a, v, x, y, \mathcal{R}\}$, where $\mathcal{R}$ produces, each time it appears in an initial program, a random floating point number between -1 and $+1$. Their set of functions was $\{+, -, *, /, exp, iflte\}$. Koza, Rice, & Roughgarden presumably constructed this set of functions and terminals via intelligent guesswork. For more details about how the initial population is generated, see (41).

The fitness of each program in the population was calculated by simulating the program with several different values for a, v, x, y and measuring the total number of insects that were eaten over the different simulations. Once the fitness of each individual program has been calculated, some fraction of the highest fitness individuals form a new population via copying themselves directly or by crossing over to create offspring. A crossover between two programs is illustrated in Figure 3. Two parents are selected to cross over, a random point is chosen in each program, and the subtrees at that point are exchanged to form two offspring, which are added to the new population. The copying and crossover procedures are repeated until a new population of M individuals has been formed. This whole process is repeated for some number G of generations. In Koza, Rice, & Roughgarden's algorithm, $M = 1000$ and $G = 61$.

Although the evolved strategies themselves were often hard to interpret, nonetheless, runs exhibited a sequence of progressively improved strategies. Each program's behavior can be visualized on a number of cases via "switching curves"—curves that illustrate in what regions respectively the lizard will chase or ignore insects. Figures 1(*b*)–(*d*) give switching curves for the best individuals in the population at generations 0, 12, and 46. It can be seen that genetic programming produced individuals with increasingly fit behavior over the course of evolution. For example, the best individual at generation 46 will avoid insects in an area that approximates region I, chase insects in a region that approximates region II, and in region III the distance the lizard is willing to travel is greatest on the x-axis and decreases with angular distance on the y axis.

In short, Koza, Rice, & Roughgarden's work showed that genetic programming can evolve increasingly fit foraging behavior in this particular simplified model. The evolved strategies can be considered hypotheses about real-life foraging

strategies—possibly difficult for humans to devise—and experiments in the real world can test their validity.

Hosts and Parasites: Using GAs to Evolve Sorting Networks

Our second example of a scientific application is work by Hillis on "host-parasite" coevolution as applied to genetic algorithms (29). Hillis, like many other GA researchers, found that in genetic algorithms, adaptation in a static environment results in both the loss of diversity in a population of candidate solutions and evolved solutions that are "overfit" to that static environment—that is, solutions that do not generalize well when placed in new environments. His solution was to have the environment itself—in the form of "parasites"—evolve to be increasingly challenging for the evolving candidate solutions.

The problem Hillis tackled was that of evolving minimal sorting networks. Sorting is a much studied problem in computer science whose goal is to place the elements in a data structure in some specified order (e.g., numerical or alphabetic) in minimal time. One particular approach to sorting is the sorting network, a parallelizable device for sorting lists with a fixed number n of elements. In a simplified form, a sorting network consists of an ordered list of comparisons to be made between elements in the given list; the compared elements are to be swapped if they are out of order. For example, the sorting network

$$(2, 5), (4, 2), (7, 14) \ldots$$

specifies that the second and fifth elements are to be compared (and possibly swapped), then the fourth and second elements are to be compared, followed by the seventh and fourteenth, and so on. A correct sorting network will take any list of a fixed length n and, after performing the specified comparisons and swaps, return the list in correctly sorted order.

In the 1960s several researchers had worked on the problem of finding correct sorting networks for $n = 16$ with a minimal number of comparisons. It was first believed that the minimum was 65 comparisons, but then smaller and smaller networks were discovered, culminating in a 60-comparison sorter. No proof of its minimality was found, but no smaller network was discovered. (See 39 for a discussion of this history).

Hillis used a form of the genetic algorithm to search for minimal sorting networks for $n = 16$. There were two criteria for networks in the population: correctness and small size. Small size was rewarded implicitly due to a diploid encoding scheme in which networks with fewer comparisons were encoded as chromosomes with more homozygous sites; smaller networks were more robust to crossovers and thus tended to be implicitly favored. Correctness was rewarded explicitly via the fitness function: Each network was tested on a sample of fitness cases (lists to be sorted). There were too many possible input cases to test each network exhaustively, so at each generation each network was tested on a set of cases chosen at random. The fitness of a network was equal to the percentage of cases it sorted correctly.

Hillis's GA was a considerably modified version of the simple GA described above. The individuals in the population were placed on a two-dimensional lattice; thus, unlike in the simple GA, there was a notion of spatial distance between two strings. Hillis hoped that this scheme would foster "speciation"—that different types of networks would arise at different spatial locations—rather than having the whole population converge to a set of very similar networks.

Nonetheless, the GA got stuck at local optima. The GA found a number of moderately good (65-comparison) solutions but could not proceed to correct smaller solutions. One reason was that after early generations the randomly generated test cases used to compute the fitness of each individual were not challenging enough. The GA had evolved strategies that worked well on the test cases they were presented with, and the difficulty of the test cases remained roughly the same. Thus, after the early generations there was no pressure on the evolving population to change the current suboptimal sorting strategies.

To solve this problem, Hillis used a form of host-parasite (or predator-prey) coevolution, in which the sorting networks were viewed as hosts and the test cases (lists of 16 numbers) as parasites. Hillis modified the system so that a population of networks coevolved on the same grid as a population of parasites, where a parasite consisted of a set of 10–20 test cases. Both populations evolved under a GA. The fitness of a network was now determined by the parasite located at the network's grid location. The network's fitness was the percentage of test cases in the parasite that it sorted correctly. The fitness of the parasite was the percentage of its test cases that the network sorted incorrectly.

The evolving population of test cases was thought to provide increasing challenges to the evolving population of networks. As the networks got better and better at sorting the test cases, the test cases presumably got harder and harder, evolving to specifically target weaknesses in the networks. This forced the population of networks to keep changing—to keep discovering new sorting strategies—rather than staying stuck at the same suboptimal strategies. With coevolution, the GA discovered correct networks with only 61 comparisons—a real improvement over the best networks discovered without coevolution, though a single comparison away from rivaling the smallest known network for $n = 16$.

Hillis's work is important because it introduced a new, potentially very useful GA technique inspired by coevolution in biology, and his results are a convincing example of the potential power of such biological inspiration. Additional work on coevolution in genetic algorithms has been done by a number of people; see, e.g., (35, 53, 62).

Educational Applications of Evolutionary Computation

It is sobering to reflect that polls show 40% of all Americans do not believe in Darwinian evolution and that this is true for 25% of college-educated Americans as well. Several factors contribute to this deficiency, including religious convictions and a lack of understanding by students and teachers alike. The mode of

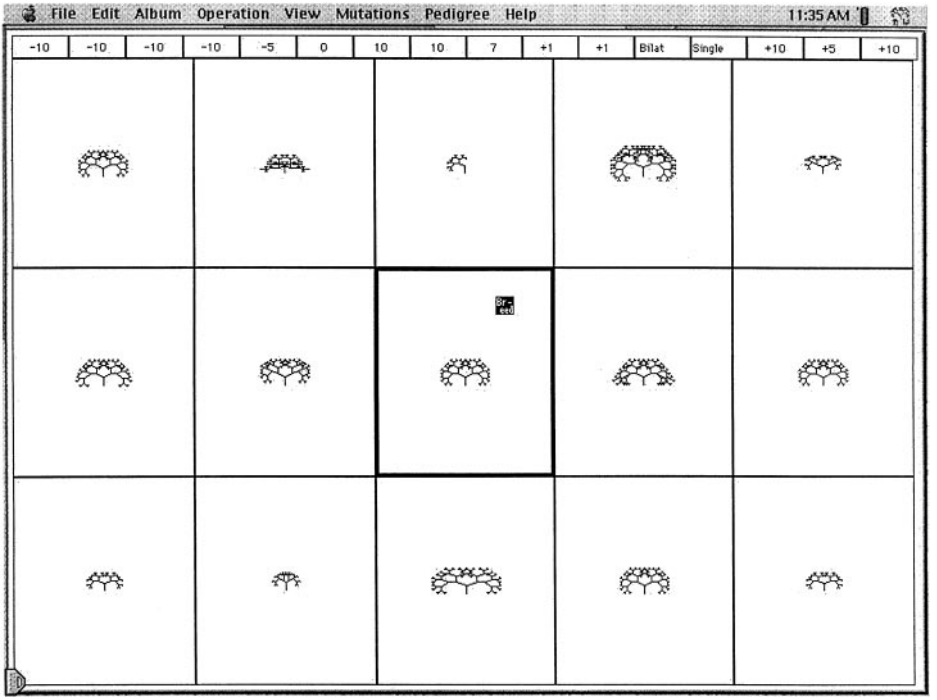

Figure 4 "Breeding screen" of the Blind Watchmaker program. The parent biomorph is present at the center of the screen and offspring biomorphs are surrounding it. Figures in the line immediately above the screen signify the genotype of the parent. Offspring biomorphs differ from the parent by mutations in the genotype string occurring with a probability determined by a parameter in the parent. Modified from (17).

presentation is also important. It is well-established that students are less likely to understand when they are passively presented with facts than when they are engaged in experimentation and construction. Several educational computer programs use evolutionary computation to address this challenge.

One such program, The Blind Watchmaker (17), is an exemplary means to teach students how evolution works. Figure 4 shows a typical screen when running the program. Near the middle is an individual "biomorph" that was chosen from the prior generation and was the "seed" for the current generation. It is surrounded by variant biomorphs that were generated by mutations. They may involve mutations of morphological features such as branch length, symmetry, gradients of length, number of branches, or even of the mutation rates and size themselves. The values for these parameters are shown in the line of figures just above the gallery of biomorphs. The figure illustrates what sorts of changes are possible even within one generation. The user can chose any of the variants shown by clicking on it, then a new generation is generated, based on variants of that biomorph, and displayed

around it. It becomes an engaging challenge to evolve forms that are exotic in one way or another, and while doing so the user acquires a real feeling for mutation, selection, and the enormous variety of phenotypes that evolution can produce. The program is keyed to Dawkins' popular book with the same title (18), which uses the program to illustrate many key features of evolution.

Like the biomorphs, the Blind Watchmaker program has itself been the seed for several interesting variants. One such variant, where the focus is on developing interesting pictures, was developed by Karl Sims and displayed at the Centre Georges Pompidou in Paris. Additional variants of this evolution have contributed to his stunning videos, such as Liquid Selves, some of which are described in (69). Besides the programs described here, there is a large variety of other popular software that employs and illustrates the creativity of evolution, including SimLife (37) and Creatures (15), nurturing hope that evolutionary computation will provide a useful teaching experience and that the next generation of American students might show a better understanding of evolution than this one has.

RELEVANCE TO PROBLEMS IN EVOLUTIONARY BIOLOGY

There is a major intellectual divide in the field of evolutionary computation, much as there is in the field of artificial intelligence. On the one hand there is a desire to make systems as efficient as possible for solving engineering problems, irrespective of any relevance to the biological world. The commercial applications discussed above, for example, frequently involve populations of 100 or so candidate solutions per generation with mutation rates on the order of 10^{-2}. Such conditions are rarely if ever met in the biological world. On the other hand, a great deal of work is now directed toward understanding general features of complex adaptive systems, including organic and cultural evolution, learning, immune reactions, and so on. The hope has been that by studying simple systems that can be dissected and rerun, it will be possible to identify rules governing adaptation that might otherwise be difficult to analyze. This pertains especially to systems with several levels of organization or that involve collective action giving rise to emergent behavior. Most of this research has been directed toward understanding the trade-offs among two or more modes of adaptation, such as learning versus evolution.

Here we can give only a sample of the biologically relevant problems that have been addressed, but we hope it gives a feel for the variety of problems, approaches, and findings that the field encompasses. We begin by discussing evolution in systems that probably do not exist in the biological world right now but that exhibit adaptation nonetheless—systems with Lamarckian evolution. (They are surprisingly effective at solving some problems.) We next discuss an interaction between two modes of adaptation—development and evolution—variously described as "genetic assimilation" or the "Baldwin effect." The Baldwin effect is thought to be important for adaptation in many artificial systems, though it has received only

scant attention by biologists. This is followed by a discussion of cultural and social evolution. These have proven difficult to study by traditional means, and the findings from evolutionary computation are often quite different from those in more traditional evolutionary biology. Finally we address some of the findings from attempts to optimize the response to evolution itself, such as optimizing the mutation or recombination rates that typically fall under the heading "evolution of genetic systems."

Lamarckian Evolution

Lamarckian Evolution refers to the evolution of traits that are modified through experience and are passed on, in their modified form, to the genotype of the next generation (42, 43). While this is consistent with certain pre-Mendelian theories of inheritance, including that which Darwin himself used, it is now recognized never to occur due to the lack of a mechanism for accomplishing it in natural systems. Artificial organisms are, of course, not subject to such constraints, and the study of Lamarckian evolution in such systems sheds some light on issues of general evolvability. For Lamarckian evolution to occur requires both a means of adapting within a generation (e.g., via development or learning) and a means of passing those gains to the genotype of the subsequent generation. Models of learning studied in this context include neural networks (63, 64), Hopfield networks (34) and production systems (28).

Hopfield networks have the ability to learn associations and, most remarkably, exhibit content addressable memory. The mere smell of a cookie, for example, might evoke all sorts of memories that have nothing to do with cookies themselves; sensing just a few properties of an object can recover a whole host of other properties. Starting from random configurations of a Hopfield network, the number of memories reliably learned and stored is approximately 0.15 times the number of nodes in a completely connected system (32). This result depends on the starting conditions, and some configurations can lead to much greater ability to remember. Imada & Araki (34) presented a set of inputs to a population of Hopfield networks capable of learning connection weights (encoded as real-valued vectors) via Hebbian (reinforcement) learning to perform a pattern-recognition task. Each generation there were learning trials, where the vectors in the evolving population were modified via a supervised-learning method. At the end of several learning trials, the weights modified via learning replaced those that had started the generation. Thus evolution was Lamarckian. Then all possible inputs were presented, and it was observed how many stable fixed points (memories) were reached by the system. If exactly those states corresponding to the inputs were obtained, then the fitness was set to its maximum value. If different vectors or more vectors were observed as fixed points, then fitnesses were diminished accordingly. After this fitness evaluation, mutation and recombination occurred, and a next generation was formed, followed by another round of learning, fitness evaluation, and selection. Nearly twice as many memories could be reliably acquired with Lamarckian

evolution as could be acquired without the Hebbian learning phase. Further, this larger number of memories could be learned even more rapidly than the much smaller number of memories acquired by networks evolving in a purely Darwinian fashion. Clearly the ability to learn and to pass this experience on through the genotype accelerated evolution.

Exploring this in more depth, Tokoro and coworkers (63, 64) found that adaptation by Lamarckian evolution was indeed much faster for neural networks than Darwinian evolution when the vectors to be learned were the same from generation to generation, that is, when the environment was constant. But when the environment changed randomly from generation to generation, then Darwinian evolution was superior. Further, when modifier genes that determined the amount of Lamarckian abilities the networks possess were themselves allowed to evolve, the Lamarckian abilities were lost completely in a randomly changing environment.

Apparently the relative advantages of Lamarckian versus Darwinian evolution alone must depend on the degree of correlation in the environment from one generation to the next, in much the way that modifiers of recombination and sexuality do. In view of the very large differences in adaptability observed here, we must expect that these differences will be likely to be exploited in practical applications of evolutionary computation.

Baldwin Effect

Lamarckian evolution is often more effective than Darwinian evolution because the space of phenotypes can be searched more extensively when each individual can try out not just one phenotype but a whole suite of new possibilities serially within their lifetime, perhaps even guided by learning. For example, in the experiments on the evolution of pattern recognition by Hopfield networks just described, each individual instantiated a genotype that generation. Under Darwinian evolution alone, a total of *number_of_agents* $*$ *number_of_generations* networks can be explored, maximum. But with learning, each trial during the Hebbian learning phase could explore yet another network, so the maximum now is *number_of_agents* $*$ *number_of_generations* $*$ *number_of_trials*, which is potentially much greater. The problem is how to pass on successful discoveries to the next generation. As is well-known, the lack of a suitable mechanism prevents biological organisms from exploiting Larmarckian evolution. There is, however, a reasonable alternative that is both possible and well-suited for evolution. This is "genetic assimilation" or the "Baldwin effect."

Many years ago C. Waddington (74) observed that *Drosophila melanogaster* normally produce posterior cross-veins in their wings. But when exposed to heat shock as pupae, they occasionally fail to develop the cross-veins in their wings when they become adults. Waddington started selection from a base population in which all of the adults had cross-veins. Each generation he heat-shocked the offspring and selected from those who were later cross-veinless as adults. After 14 generations, 80% of those who were heat-shocked were cross-veinless, and a

few began to be cross-veinless even without the shock. With subsequent selection he obtained lines with as many as 95% cross-veinless in the absence of shock. He recognized that this was not Lamarckian evolution, but that it rather resulted simply from changing the thresholds for expression of the cross-vein trait; Waddington termed the phenomenon "genetic assimilation" (74). It also happened that a similar phenomenon had been described earlier by JM Baldwin and is sometimes called the "Baldwin effect" (10). In textbooks of evolution this phenomenon is occasionally mentioned but seldom receives more than a brief note.

The Baldwin effect has been observed in evolutionary computation studies (see, e.g., 1, 30). In Waddinton's study the problem was to select for a trait (cross-veinlessness) that is almost never expressed. The importance for evolutionary computation is slightly different; it sometimes occurs that a trait is enormously useful if it is fully developed or expressed, but it is of no use otherwise. The problem is to hit upon the right (and rare) configuration of alleles, then preserve it for further selection and elaboration. In an asexual population, the right ensemble of alleles might never (or almost never) arise. In a sexual population it might arise but would tend to be broken up immediately by recombination. However, if learning or other forms of adaptation during individuals' lifetime are available, the desired configuration can arise via these mechanisms; while the trait itself will not be passed on to offspring, the genetic background producing it will be favored. Thus, according to Baldwin, learning and other forms of within-lifetime adaptation can lead to increased survival, which can eventually lead to genetic variation that produces the trait genetically.

This effect has been demonstrated in simple evolutionary computation settings. For example, Hinton & Nowlan (30) considered neural networks that evolved via GAs. At each generation every individual in the population had a "lifetime" during which its weights were learned. Each weight was coded by a different locus in the network's genome. The alleles at each locus were 0, 1, or ?, where "?" signified that the value varied with learning, and where "learning" consisted of a series of trials in which the ? values were guessed to be 0 or 1. A final weight of value 1 came either from having the "1" allele in one's genome or from having adopted it in a guessing trial. Populations of networks evolved under a fitness function that highly rewarded networks when all connections were set to 1 sometime during the network's lifetime but not otherwise. If the 1 state was adopted early in a network's lifetime, then the fitness was higher than if it was adopted later. With this combination of evolution and learning, Hinton & Nowlan observed that correct settings for all loci were achieved after about 20 generations of the GA, whereas they never occurred under evolution alone. Hinton & Nowlan interpreted this result as an (extremely simple) example of the Baldwin Effect. Maynard Smith (46) calculated that if phenotypes were strictly determined by genotype, without opportunity for learning, then about 1000 generations would have been required in an asexual population and would probably never evolve in a strictly sexual population. As described by Hinton & Nowlan (30), learning makes the difference between finding a needle in a haystack and finding a needle in the haystack when someone tells you

when you are getting closer. It is evident from these studies, both with Lamarckian evolution and the Baldwin effect, that learning often allows organisms to evolve *much* faster than their non learning equivalents. Hence its importance for tracking the stock market by adapting agents, and quite possibly for evolution in the natural world.

Cultural and Social Evolution

In view of the effect of learning on evolution, it is natural to ask how culture affects evolution. It would seem that artificial systems like those used for studying Lamarckian evolution and the Baldwin effect would be natural vehicles to explore the elements of cultural transmission. But in fact, there have been relatively few such studies.

Studies of the evolution of cooperation in the Prisoner's dilemma, begun by Axelrod (5), have stimulated a great deal of investigation. These typically do not involve cultural evolution, though in the real world such traits would have a very strong cultural component. There have been a few studies on the evolution of communication: acquiring the ability to communicate and agree on common lexical items have been modeled with some success (4, 70, 75). In addition, a few studies have addressed the very difficult problems concerned with how actual languages are learned and evolve (see 51).

Learning human languages presents serious theoretical problems for complex adaptive systems. For example, Gold's problem (25) is concerned with how, after hearing only a finite number of sentences (many of which may have errors), each of us learns a grammar that can generate an infinite number of grammatically correct sentences. A second problem is to account for the many changes that occur through time. Speakers of modern English can typically read Shakespeare (early Modern English from 400 years ago) without much difficulty. We can read Chaucer (Middle English from 800 years ago) with a moderate amount of lexical help. Only scholars can read Bede (Old English from 1300 years ago). Spoken Chaucer would be incomprehensible because of changes in vowel sounds that have occurred since it was spoken, but the spelling has remained similar. The challenge is to describe and, possibly predict, the course of language evolution in cases such as this. Learnability is a major issue here, and it is generally felt that the evolution of languages is largely driven by how easy it is to learn. Niyogi & Berwick (51) have used evolutionary computation methods to model how populations of agents can acquire language from hearing samples from the prior generation and then themselves provide examples for the next generation to learn. Using Chomsky's principles and parameter model of language (12), they found that some parameters were more stable than others. Further, they found that learnability alone was an inadequate explanation for some of the changes in grammatical form known to have occurred in the evolution of English, such as verb order, where Old English resembles present-day German. For a review of other attempts to model the coevolution of genes, cultures, and societies see Epstein & Axtell (20).

Evolution of Computational Ecologies

A novel approach to computational studies of ecologies was pioneered by T Ray (59). His goal was to evolve self-replicating, cooperating bits of computer code that "live" in a virtual computer world. Most computer instructions are too brittle to permit random mutations and recombinations that will provide fragments of code that are both meaningful and capable of being strung together with other such fragments. Koza's genetic programming paradigm, described in the section on using Genetic Programming to Evolve Optimal Foraging Strategies, is one method to get around this brittleness; neural networks are another. Ray's "Tierra" program uses a different method involving a specially designed assembly language to construct self-replicating programs. The resulting "ecology" provides interesting parallels to natural life—including competition for (memory) resources, trophic structures, and so on (47). Ray's current efforts are directed toward the evolution of self-contained but cooperating programs that emerge through evolutionary computation and are analogous to multicellular organisms. Success has so far been limited, but Ray does observe differentiation into something akin to somatic and reproductive code. (60).

Building on Tierra, C Adami (2) has developed a computational world, "Avida," with spatial structure that Tierra lacks. J Chu (14) has further developed Avida to run on the massively parallel Intel Paragon computer, so that very large numbers of simulations can be run in fairly large environments, e.g. 100×100 units, with as many as 10,000 competing bits of code. Chu observed what seem to be invariant power laws, where the log of the number of copies of a program that have "lived" can be plotted against how frequently such sequences were observed in the evolutionary sequence. When selection was strong he found a $-3/2$ slope for this, just as is observed for the number of families in the fossil record (67) and is observed for avalanche size in the higher-dimension sandpiles of Bak's models of self-organized criticality (9). Chu developed arguments based on the theory of branching process to explain why this should be true. Such relationships, if found to be general, might point to a radically different theory of evolution than we now have, based on principles of self-organizing systems that are both more general and also more capable, in that they can capture phenomena that have so far resisted adequate explanation (19).

Adaptive Surfaces and Evolution of Genetic Systems

Until recently, the field of evolutionary computation has made surprisingly little use of quantitative genetics or population genetics theory. One reason for this is the belief by computer scientists that most difficult problems are highly nonlinear—that is there exists widespread interaction among the parts of a candidate solution, so that attempts to minimize interactions (as by attempts to linearize in quantitative genetics) or to treat genes as individuals (in single-locus models of population genetics) are bound to be self-limiting. A second reason for this lack of reciprocity in theory is that the conditions for evolution are often different in evolutionary

computation settings (e.g., population sizes of a few hundred and mutation rates of 10^{-2}) than in biology. Further, while most evolutionary computation systems include recombination, the life cycle of individuals is like that of a moss, with a short diploid and a long haploid phase—not at all what most genetic theory addresses.

This is not to say that population genetics is inconsistent or inapplicable. Christiansen & Feldman (13) showed how to derive parts of Holland's GA theory (31) from principles of population genetics. Further, theoretical predictions from population genetics do help explain certain observations of evolutionary computation: e.g. in evolutionary computation applications where mutation rates and magnitudes are allowed to evolve, it is typically observed that they evolve downwards after sufficient time (reviewed in 8), as expected from equilibrium theory in population genetics.

One of the most challenging problems in population genetics has concerned the manner that populations traverse their adaptive landscape. Does evolution carry out a gradual hill-climbing, leading to some sort of optimization, as RA Fisher argued, or does it proceed by jumps and starts, with chance playing a significant role, as argued by Sewall Wright? In spite of the centrality of this issue for many questions in evolutionary theory, it has proven extremely difficult to test different proposals (16). Evolutionary computation has addressed this problem from a purely practical standpoint and has typically found that population subdivision ("island models") significantly speeds evolution (e.g., 8, 27, 71). From a different vantage, theoretical approaches to evolutionary computation, such as those proposing mechanisms underlying metastability in evolution (72, 73), may provide new theoretical bases for describing many of these phenomena.

One feature of adaptive landscapes, in both evolutionary computation and biological settings, is that broad plateaus of fitness seem common, and chance plays a major role in moving about on them. Where the population can move next seems to depend critically upon where it has drifted on the plateau. For example, Huynen, Stadler & Fontana (33) used computational models for predicting molecular structures to describe the 3D structure of tRNA. While 30% of nucleotide substitutions seemed to be neutral, the high dimensionality made it possible to traverse sequence space along a connected path, changing every nucleotide present, without ever changing the structure. It is no surprise then, that when a population is begun with all sequences identical, but with a small amount of mutation, the initial point in sequence space diffuses outward into a cloud, to a limit in accord with theoretical expectations, then drifts along the plateau. Different subpopulations can reach very different parts of the sequence space before dramatic improvement results from finding one improvement or another. Fitness assumes a step-like improvement, coinciding with Wright's expectation that "Changes in wholly nonfunctional parts of the molecule would be the most frequent ones but would be unimportant, unless they occasionally give a basis for later changes which improve function in the species in question, which would then become established by selection" (56), p. 474.

Moving from plateau to plateau of fitness is frequently observed in evolutionary computation and is typically associated with changes in complexity. Taking just one example, Miglino, Nafisi, & Taylor (48) used genetic algorithms to evolve controllers for small robots. The controllers were neural networks that could be described by equivalent finite state automata, the complexity of which can be readily observed and measured. The task presented was to traverse as many squares as possible in a grid placed on the floor in a limited amount of time. Starting from random networks it was initially sufficient merely to move forward and turn left when encountering a corner. Many neural networks prescribed this behavior, but some of these made it easier to make jumps to radically more sophisticated behavior, with correspondingly more complex programs. There was much variation from run to run, with chance largely determining which populations were able to find one improved solution or another.

While these studies showed quite clearly that plateaus on the adaptive surface are common, with stepwise improvement in fitness, it must be stressed that this is not always the case—especially when fitnesses are frequency-dependent. Very complex dynamics are sometimes observed, including plateaus interspersed with periods of chaos (45). An interesting example of this is provided by competition among bit strings in a series of studies by K Kaneko and co-workers (summarized in 36). In this system strings are assumed to compete to the extent that they are similar (measured by their Hamming distance)—more similar strings compete more strongly, so fitness is frequency-dependent. But strings too far apart have less success in reproduction. Mutation among the strings is allowed. It is also possible to include predator/prey interactions in this system, where the strength of predator-prey interactions depends on the Hamming distance. Such systems are high-dimensional and highly nonlinear. In a way, their interactions resemble logistic maps, which are known to be chaotic over much of their parameter space, except that here they are high-dimensional and can escape from having their fitness reduced by competition, as it were, through mutation to a less frequent form. In a series of papers Kaneko and co-workers analyzed the dynamics of this system, numerically calculating the Lyapunov exponents, and observed high-dimensional, weakly chaotic dynamics in the evolution of this system that often led to dynamic stability and robustness against external perturbations. He termed this situation "homeochaos" and suggested that such system dynamics may be very general features of evolution, both in computational evolution and in the real world.

CONCLUSIONS

There are many parallels between biological evolution searching through a space of gene sequences and computer evolution searching through a space of computer programs or other data structures. Several approaches to exploit these similarities have developed independently and are collectively termed evolutionary algorithms or evolutionary computation. Such methods of computation can be used to search

through the space of candidate solutions to problems and are now finding application in an increasing number of industrial and business problems.

While there is no general theory that will identify the best method to find optima, it appears that evolutionary computation is best suited for problems that involve nonlinear interactions among many elements, where many intermediate optima exist, and where solutions that are satisficing—merely very good without necessarily being the absolute optimum—will do. Such problems are common in business and in biological studies, such as cooperation, foraging, and coevolution.

Evolutionary computation can sometimes serve as a useful model for biological evolution. It allows dissection and repetition in ways that biological evolution does not. Computational evolution can be a useful tool for education and is beginning to provide new ways to view patterns in evolution, such as power laws and descriptions of non-equilibrium systems. Evolutionary theory, as developed by biologists, typically tries to linearize systems, for ease of analysis with differential equations, or to treat units in isolation, as in single-locus selection. While evolutionary computation is not inconsistent with such theory, it tends to be outside it, in that real difference in capacity and complexity are often observed and are not really describable by stable equilibria or simple changes in gene frequencies, at least in ways that are interesting. There is reason to believe that theories of evolutionary computation might extend the language of biological evolutionary theory and contribute to new kinds of generalizations and analyses that have not been available up to now.

ACKNOWLEDGMENTS

MM acknowledges the Santa Fe Institute and the National Science Foundation (grant NSF-IRI-9705830) for support. CT acknowledges NSF grant #5BR9720410.

LITERATURE CITED

1. Ackley D, Littman M. 1992. Interactions between learning and evolution. In *Artificial Life II*, ed. CG Langton, C Taylor, JD Farmer, S Rasmussen, pp. 487–509. Reading, MA: Addison-Wesley
2. Adami C. 1998. *Introduction to Artificial Life*. New York: Springer-Verlag
3. Angeline PJ, ed. 1997. *Evolutionary Programming VI: 6th Int. Conf. EP97*. New York: Springer
4. Arita T, Koyama Y. 1998. Evolution of linguistic diversity in a simple communication system. In *Artificial Life VI*, ed. C Adami, RK Belew, H Kitano, CE Taylor, pp. 9–17. Cambridge, MA: MIT Press
5. Axelrod R. 1984. *The Evolution of Cooperation.* New York: Basic
6. Bäck T. 1996. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford: Oxford Univ. Press
7. Bäck T, ed. 1997. *Proceedings of the Seventh International Conference on Genetic Algorithms*, San Francisco, CA: M. Kaufmann
8. Baeck T, Hammel U, Schwefel HP. 1997. Evolutionary computation: comments on

the history and current state. *IEEE Trans. Evol. Computation* 1:3–17
9. Bak P. 1996. *How Nature Works: The Science of Self-Organized Criticality.* New York: Springer-Verlag
10. Belew RK, Mitchell M, eds. 1996. *Adaptive Individuals in Evolving Populations: Models and Algorithms.* Reading, MA: Addison Wesley
11. Belew RK, Vose MD, eds. 1997. *Foundations of Genetic Algorithms 4.* San Francisco, CA: M. Kaufmann
12. Chomsky N. 1995. *The Minimalist Program.* Cambridge, MA: MIT Press
13. Christiansen FB, Feldman MW. 1998. Algorithms, genetics, and populations: the schemata theorem revisited. *Complexity* 3(3):57–64
14. Chu J. 1999. *Computational explorations of life.* PhD thesis. Calif. Inst. Technol., Pasadena, CA
15. Cliff D, Grand S. 1999. The 'Creatures' global digital ecosystem. *Artificial Life.* In press
16. Coyne JA, Barton N, Turelli M. 1997. Perspective: a critique of Sewall Wright's shifting balance theory of evolution. *Evolution* 51:643–71
17. Dawkins R. 1989. The evolution of evolvability. In *Artificial Life*, ed. CG Langton, 201–220. Reading, MA: Addison-Wesley
18. Dawkins R. 1996. *The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe Without Design.* New York: Norton. 2nd ed.
19. Depew DJ, Weber BH. 1995. *Darwinism Evolving.* Cambridge, MA: MIT Press
20. Epstein J, Axtell R. 1996. *Growing Artificial Societies.* Cambridge, MA: MIT Press
21. Fogel DB. 1995. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* New York: IEEE Press
22. Fogel DB, ed. 1998. *Evolutionary Computation: The Fossil Record.* New York: IEEE Press
23. Fogel LJ, Owens AJ, Walsh MJ. 1966. *Artificial Intelligence Through Simulated Evolution.* New York: John Wiley
24. Gehlhaar D, Verkhivker G, Rejto P, Sherman C, Fogel D, et al. 1995. Molecular recognition of the inhibitor AG-1343 by HIV-1 protease: conformationally flexible docking by evolutionary programming. *Chem. Biol.* 2:317–24
25. Gold EM. 1967. Language identification in the limit. *Inform. Control* 10:447–74
26. Goldberg DE. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley
27. Gordon VS, Whitley D. 1993. Serial and parallel genetic algorithms as function optimizers. In *Proc. Fifth Int. Conf. Genetic Algorithms*, ed. T Bäck, pp. 177–183. San Mateo, CA: M. Kaufmann
28. Grefenstette JJ. 1991. Lamarckian learning in multi-agent environments. In *Proc. 4th Int. Conf. on Genetic Algorithms and Their Applications*, ed. RK Belew, L Booker, pp. 303–10. San Mateo, CA: M. Kaufmann
29. Hillis WD. 1990. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D* 42:228–34
30. Hinton GE, Nowlan SJ. 1987. How learning can guide evolution. *Complex Systems* 1:495–502
31. Holland JH. 1975. *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: Univ. Michi. Press
32. Hopfield J. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proc. Nat. Acad. Sci. USA* 79:2554–58
33. Huynen MA, Stadler F, Fontana W. 1996. Smoothness within a rugged landscape: The role of neutrality in evolution. *Proc. Natl. Acad. Sci. USA* 93:397–401
34. Imada A, Araki K. 1996. Lamarckian evolution and associative memory. In *Proc. 1996 IEEE Third Int. Conf. Evol. Computation (ICES-96)*:676–80
35. Juillé H, Pollack JB. 1998. Coevolutionary learning: a case study. In *ICML '98-Proc.*

Int. Conf. Machine Learning. San Francisco, CA: M. Kaufmann
36. Kaneko K. 1994. Chaos as a source of complexity and diversity in evolution. *Artificial Life* 1:163–77
37. Karakotsios K, Bremer M. 1993. *SimLife: The Official Strategy Guide.* Rocklin, CA: Prima
38. Kelly K. 1994. *Out of Control: The Rise of Neo. Biological Civilization.* Reading, MA: Addison-Wesley
39. Knuth DE. 1973. *The Art of Computer Programming.* Vol. 3: *Sorting and Searching.* Reading, MA: Addison-Wesley
40. Koza JR. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: MIT Press
41. Koza JR, Rice JP, Roughgarden J. 1992. Evolution of food foraging strategies for the Caribbean anolis lizard using genetic programming. *Adaptive Behav.* 1(2):47–74
42. Lamarck JB. 1809. *Philosophie Zoologique, ou Exposition des Considérations Relatives a l'Histoire Naturelle de Animaux.* Paris: Chez Dentu et L'Auteur
43. Lamarck JB. 1996. Of the influence of the environment on the activities and habits of animals, and the influence of the activities and habits of these living bodies in modifying their organization and structure. See Ref. 10, pp. 39–57
44. Lewontin R. 1998. Survival of the nicest. *NY Rev. Books.* 22 Oct. 1998, 59–63
45. Lindgren K. 1992. Evolutionary phenomena in simple dynamics. In *Artificial Life II*, ed. CG Langton, C Taylor, JD Farmer, S Rasmussen, pp. 295–312. Reading, MA: Addison-Wesley
46. Maynard Smith J. 1987. Natural selection: when learning guides evolution. *Nature* 329:761–62
47. Maynard Smith J. 1992. Byte-sized evolution. *Nature* 355:772–73
48. Miglino O, Nafasi K, Taylor CE. 1994. Selection for wandering behavior in a small robot. *Artificial Life* 2:101–16
49. Mitchell M. 1996. *An Introduction to Genetic Algorithms.* Cambridge, MA: MIT Press
50. Murakawa M, Yoshizawa S, Adachi T, Suzuki S, Takasuka K, et al. 1998. Analogue EHW chip for intermediate frequency filters. In *Evolvable Systems: From Biology to Hardware*, ed. M Sipper, D Mange, pp. 134–43. New York: Springer
51. Niyogi P, Berwick RC. 1995. *The Logical Problem of Language Change. Tech. Rep. A. I. Memo No. 1516.* MIT Artificial Intelligence Lab. Cambridge, MA
52. Papadimtriou CH, Sideri M. 1998. On the evolution of easy instances. *Unpublished manuscript*, Computer Science Dept., University of California, Berkeley, CA
53. Paredis J. 1997. Coevolving cellular automata: Be aware of the red queen! In *Proc. Seventh Int. Conf. Genetic Algorithms*, ed. T Bäck, pp. 393–400. San Francisco, CA: Morgan Kaufmann
54. Petzinger Jr. T. 1995. At Deere they know a mad scientist may be the firm's biggest asset. *Wall Street J.* 14 July 1995, p. A1
55. Press WH, A. Teukolsky S, Vetterling WT, Flannery BP. 1992. *Numerical Recipes in C.* New York: Cambridge Univ. Press
56. Provine WB. 1986. *Sewall Wright and Evolutionary Biology.* Chicago, IL: Univ. Chicago Press
57. Rao SS. 1998. Evolution at warp speed. *Forbes Mag.*
58. Rawlins G, ed. 1991. *Foundations of Genetic Algorithms.* San Mateo, CA: M. Kaufmann
59. Ray TS. 1991. An approach to the synthesis of life. In *Artifical Life II*, ed. CG Langton, C Taylor, J Farmer, S Rasmussen, pp. 371–408. Reading, MA: Addison–Wesley
60. Ray TS, Hart J. 1998. Evolution of differentiated multi-threaded digital organisms. In *Artificial Life VI*, ed. C Adami, RK Belew, H Kitano, CE Taylor, pp. 295–306. Cambridge, MA: MIT Press
61. Rechenberg I. 1973. *Evolutionsstrategie.* Stuttgart: Frommann-Holzboog

62. Rosin CD, Belew RK. 1995. Methods for competitive coevolution: finding opponents worth beating. In *Proc. Sixth Int. Conf. Genetic Algorithms*, ed. LJ Eshelman, pp. San Francisco, CA: M. Kaufmann
63. Sasaki T, Tokoro M. 1997. Adaptation toward changing environments: Why Darwinian in nature? In *Proc. Fourth Eur. Conf. on Artificial Life*, 145–53. Cambridge, MA: MIT Press
64. Sasaki T, Tokoro M. 1999. Evolvable learnable neural networks under changing environments with various rates of inheritance of acquired characters: comparison between Darwinian and Lamarckian evolution. *Artificial Life.* In press
65. Schwefel HP. 1975. *Evolutionsstrategie und Numerische Optimierung.* PhD thesis, Technische Univ. Berlin, Berlin
66. Schwefel HP. 1995. *Evolution and Optimum Seeking.* New York: Wiley
67. Sepkowski JJ. 1992. *A Compendium of Fossil Marine Animal Families.* Milwaukee, WI: Milwaukee Public Mus. 2nd ed.
68. Simon H. 1969. *The Sciences of the Artificial.* Cambridge, MA: MIT Press
69. Sims K. 1994. Evolving 3D morphology and behavior by competition. In *Artificial Life IV*, ed. RA Brooks, P Maes, pp. 28–39. Cambridge, MA: MIT Press
70. Steels L, Kaplan F. 1998. Stochasticity as a source of innovation in language games. In *Artificial Life VI*, ed. C Adami, RK Belew, H Kitano, CE Taylor, pp. 368–78. Cambridge, MA: MIT Press
71. Tanese R. 1989. Distributed genetic algorithms. In *Proc. Third Int. Conf. on Genetic Algorithms*, ed. JD Schaffer, pp. 434–39. San Mateo, CA: M. Kaufmann
72. van Nimwegen E, Crutchfield JP, Mitchell M. 1999. Statistical dynamics of the Royal Road genetic algorithm. *Theoret. Computer Sci.* To appear
73. van Nimwegen E, Crutchfield JP, Mitchell M. 1997. Finite populations induce metastability in evolutionary search. *Phys. Lett. A,* 229(2):144–50
74. Waddington CH. 1953. Genetic assimilation of an acquired character. *Evolution* 7:118–26
75. Werner GM, Dyer MG. 1991. Evolution of communication in artificial organisms. In *Artificial Life II*, ed. CG Langton, C Taylor, J Farmer, S Rasmussen, pp. 659–87. Reading, MA: Addison Wesley
76. Whitley LD, ed. 1993. *Foundations of Genetic Algorithms 2.* San Mateo, CA: M. Kaufmann
77. Whitley LD, Vose MD, eds. 1995. *Foundations of Genetic Algorithms 3.* San Francisco, CA: M. Kaufmann