Without Miracles

13 Evolutionary Computing: Selection Within Silicon

When man wanted to fly, he first turned to a natural example--the bird--to develop his early notions of how to accomplish this difficult task. Notable failures by Daedalus and numerous bird-like contraptions (ornithopters) at first pointed in the wrong direction, but eventually, persistence and the abstraction of the appropriate knowledge (lift over an airfoil) resulted in successful glider and powered flight. In contrast to this example, isn't it peculiar that when man has tried to build machines to think, learn, and adapt he has ignored and largely continues to ignore one of nature's most powerful examples of adaptation, genetics and natural selection?

--David Goldberg[1]

Over the last several decades the digital computer has become an important tool in a growing number of human activities. From writing school reports and managing household budgets to performing complex numerical analyses and simulating astrophysical events, the computer quickly becomes indispensable to anyone who takes the trouble to learn to use one. Its ability to store, manipulate, and analyze large amounts of data and provide stunning visual displays thereof have made it particularly useful in all fields of scientific inquiry.

Among its many scientific achievements, the computer has helped us to further our knowledge of biological evolution and the cumulative blind variation and selection processes on which adaptive evolution depends. Perhaps even more significant is that computers are now being used to solve extremely complex problems in many areas of science, engineering, and mathematics, generating solutions not imagined by the scientists who posed the problems and translated them into computer-readable form.

In this chapter we will survey some exciting new developments at the intersection of computers and evolution. In doing so, we will gain a better appreciation both of the power of the evolutionary process and how this power can be harnessed to solve complex problems--not over the centuries and millennia of geological time as in biological evolution, but over much briefer periods of time.

Recreating the Process of Evolution

Darwinian evolution remains the accepted scientific explanation for the origin and design of all the life forms on our planet; however, the slowness of biological evolution makes it impossible to observe in the way that other scientific phenomena can be studied. Solar eclipses can be observed from start to finish; cannonballs can be dropped from towers and their descents timed; and the entire life cycles of fruit flies, snakes, and rabbits can be studied. But biological evolution takes place over such long periods of time that its effects cannot readily be observed during any one human lifetime.[2] The large time scale of organic evolutionary change is no doubt a major reason for its rejection and misunderstanding by so many otherwise educated and well-informed individuals.

But computers have now made it possible to model the processes involved in biological evolution and to do this quickly enough so that they can be observed and studied over periods of time as short as hours, minutes, and even seconds. One of the first to create such simulations was Oxford evolutionary zoologist Richard Dawkins who reported his results in *The Blind Watchmaker*, his popular and colorful introduction to evolution.[3]

Dawkins's computer program is made up of two parts.[4] The first program, called DEVELOPMENT, produces on the computer screen a set of treelike line drawings called biomorphs, each representing an organism. The generation of each biomorph is based on a genome comprising nine genes, each of which can take a value from -9 to +9. For example, one gene controls width, so that a low value for this gene would result in a narrow biomorph and a higher value would produce a wider one. The remaining eight genes control other aspects of the biomorphs, such as height and the number of branchings added to the central stem. By specifying values for each of the genes, DEVELOPMENT produces the corresponding biomorph using the gene values as a sort of genetic recipe, not unlike the way in which real genes direct the development of living organisms.

But since evolution as we know it cannot occur without reproduction, this is the function of the second part of the program. REPRODUCTION takes the genes of the original single parent and passes them down to a set of 14 progeny. Although the reproduction is asexual, a rather high rate of random mutation is used so that each offspring differs from the parent on one of the nine genes. This provides the blind variation on which selection can operate.

Selection is determined by the eye of the program user. After each new litter of biomorphs is generated, the operator may select any one of the 14 mutant children to be the parent for the next generation. The criteria for this are completely up to the user. Dawkins provides a lively account of what happened when he decided to select cumulatively those biomorphs that most resembled an insect:

> When I wrote the program, I never thought it would evolve anything more than a variety of tree-like shapes. . . . Nothing in my biologist's intuition, nothing in my 20 years' experience of programming computers, and nothing in my wildest dreams, prepared me for what actually emerged on the screen. I can't remember exactly when in the sequence it first began to dawn on me that an evolved resemblance to something like an insect was possible. With wild surmise, I began to breed, generation after generation, from whichever child looked most like an insect. My incredulity grew in parallel with the evolving resemblance. . . . Admittedly they have eight legs like a spider, instead of six like an insect, but even so! I still cannot conceal from you my feeling of exultation as I first watched these exquisite creatures emerging before my eyes. I distinctly heard the triumphal opening chords of *Also sprach Zarathustra* (the 2001 theme) in my mind. I couldn't eat, and that night "my" insects swarmed behind my eyelids as I tried to sleep.[5]

Dawkins's program, although different in many ways from genuine biological evolution, dramatically demonstrates the potential of cumulative blind variation and selection. Each generation of 14 progeny is created from blind, random mutations of the parent's set of genes. Over this the human operator has no control. The operator does not create these forms, but rather they emerge from the blind variation algorithm of the computer program as part of the more than 322 billion combinations of nine genes, each having 19 different possible values. But by cumulatively selecting for certain characteristics, one can gradually discover biomorphs that resemble bats, lunar landers, foxes, scorpions, airplanes, or any of countless other possibilities (it took Dawkins only 29 generations to evolve his insect from a single point). And these myriad forms can be produced by a set of only nine genes, many orders of magnitude fewer than the millions that make up the genome of even relatively simple organisms.

As impressive as Dawkins's biomorph program is, it is quite unlike genuine biological evolution in that a human being is the sole agent of selection. This unnatural selection-by-humans has been used for hundreds of years in the selective breeding of food crops, flowers, and domesticated animals. In natural selection it is the organism's own success at surviving and reproducing that determines which variations will survive and which others will feel the blow of Darwin's hammer. And so Thomas Ray of the University of Delaware became obsessed with the idea of creating a computer environment that could simulate evolution free of any active human involvement in the selection process.[6]

In the artificial computer environment Ray calls Tierra, organisms are modeled as strings of computer code that compete with each other for memory space and processing time. Organisms that are successful in finding matching bits of code in their environment are able to reproduce, and those that are unsuccessful are eventually eliminated. Genetic variation is provided through processes designed to mimic the genetic mutations of real organisms caused by cosmic radiation and errors of replication.

Ray completed his programming of Tierra just as 1990 began, and on the night of January 3 introduced a single self-replicating organism into his silicon-based environment, or soup, as he called it. He thought at the time that this original ancestor would provide only a preliminary test of his simulator, and that it would likely take years of additional programming before a sustainable process resembling real evolution would emerge. But his rather pessimistic expectations were not to be borne out, as he recounts that "I never had to write another creature."[7]

Ray's original creature, called the Ancestor, consisted of computer code including 80 instructions. As it reproduced, its clones began to fill up the available memory space while the Ancestor and its oldest progeny began to die off. But then mutants containing 79 instructions appeared and successfully competed with the original clones. Before too long a new mutant arrived on the scene, but it had only 45 instructions, too few, Ray reckoned, for it to replicate on its own. And yet there it was, successfully competing with the much more complex creatures. Apparently this new organism was a type of parasite that had evolved the ability to borrow the parts of the necessary replication code from the more complex organisms. But as these parasites grew in numbers, they began to crowd out the host organisms on which their own reproduction depended, and so they began to die off as well. This decline in the numbers of parasites led in turn to an increase in the population of host creatures, causing an oscillatory cycle between the two types of organisms in the same way that cycles of growth and decline between hosts and parasites or predators and prey are observed in nature.

But more than just a simple moving pendulum between hosts and parasites was taking place. The hosts began to evolve characteristics that would make them resistant to the parasites, and the parasites found ways of circumventing these new defense systems. An ever-escalating evolutionary arms race was in progress, the very phenomenon that is believed to have provided the springboard for the increasing adaptive complexity of living organisms over evolutionary time.[8]

Ray also found other types of evolutionary processes occurring, such as forms of cooperation among highly related organisms, cheating, and sexual reproduction. Although he had designed both the Tierran environment and its first inhabitant, his role now switched to that of observer once this Ancestor began to reproduce, mutate, and evolve. But instead of observing only the end products of evolution as he had done during his many years of field work in the rain forests of Costa Rica, Ray was able to observe the process of evolution itself. And the new field of artificial life was finally on its way to becoming recognized by the scientific community as an important new arena for research.[9]

The work of Dawkins and Ray on simulated evolution are just two examples of the many investigations of artificial life that have been carried out.[10] Their work was among the first to show how computers could be used to simulate the same evolutionary processes responsible for all of earth's life forms. For those skeptical that the processes of cumulative blind variation and selection can produce the complex, adapted structures that make up living organisms, research in artificial evolution reports striking evidence consistent with the selectionist explanation proposed by Darwin for the origin and adaptive evolution of species over a century ago. As McGill University biologist Graham Bell noted, "Many people doubt that the theory of evolution is logically possible. . . . Now, one can simply point to the output of Ray's programs; they are the ultimate demonstration of the logical coherence of evolution by selection."[11] And although neither Dawkins nor Ray initially doubted the power of natural selection, even they were surprised to discover how easy it was to make evolution happen on a computer once the basic processes of variation, selection, and reproduction were modeled as programs.

Dawkins's and Ray's work also demonstrated the creative nature of evolution. Even though they were responsible for designing the computer environment, algorithms, and original digital organisms, the organisms that later evolved and the complex interactions among them were not explicitly programmed. Rather, they emerged in bottom-up fashion as the creative products of the evolutionary process itself. And if artificial evolution could be used to create complex line drawings and digital creatures able to survive and reproduce in a challenging silicon environment, it seemed reasonable to expect that the same process could be harnessed to solve practical, real-world problems, and perhaps to create intelligence itself.

The Computer Can Know More than the Programmer

Attempts to use evolution-based techniques to find solutions to novel problems and to develop artificial intelligence actually predate by quite some time the work of Dawkins and Ray. In 1966 Lawrence Fogel, Alvin Owens, and Michael Walsh published a slim volume in which they demonstrated how evolutionary processes implemented on a computer could be used to find solutions to problems involving prediction, diagnosis, pattern recognition, and control system design.

In their prediction experiments, these researchers started out by creating a single parent "machine" out of computer code. This machine was made up of a set of rules relating input conditions to output conditions, which was used to make the desired prediction. The prediction would be made and evaluated, after which the parent machine would produce an offspring with a slight, random mutation. The offspring's prediction would then be evaluated and compared with that of the parent's. If it was better, the offspring would be mutated to form the next machine; if the parent's prediction was better, the offspring would be discarded and the parent would be permitted to reproduce again. This process would continue until the predictions were within a desired range of error, or until some predetermined time or computing limit was reached.

This method can perhaps be better understood using the analogy of preparing a stew. You start with a recipe, perhaps one found in a cookbook, or provided by your mother or other respected culinary artist. After making up a pot of stew, you keep it warm while you make up another pot, randomly changing some part of the recipe, perhaps adding more salt or cooking it 20 minutes longer. You then compare the taste of the two pots of stew and discard the one (and recipe) you judge inferior. If the original stew was retained, another mutant stew would be prepared, tasted, and evaluated, but if the second stew was preferred it would be mutated again and compared with the new result. By repeating this procedure many times, the stew should continue to get better until any further change would only make it worse (according to your taste, that is).

Of course, for someone with knowledge of foods and cooking techniques, the outcome of a recipe will itself suggest certain changes--if the food is undercooked, cook it longer; if it is too spicy, cut back on the chili peppers. But in Fogel's method, as in biological evolution, mutations are completely blind in that the shortcomings of the current computer code provide no information about how the code should be changed to make it perform better. So for the cooking analogy to hold, we have to imagine a pure novice with absolutely no knowledge of cooking.

But this procedure is in one respect quite different from biological evolution. Whereas natural selection typically operates on a large population of organisms that vary in many different ways, this method compared only one parent and a single mutated offspring at each step. Nonetheless, this approach to evolutionary computing used a (rather limited) source of essential blind variation that, when coupled with cumulative selection, led to new solutions that had not been foreseen or in any way explicitly programmed into the simulation. And although their results were quite modest and did not have a major impact at the time on the field of artificial intelligence, Fogel and his associates were among the first to demonstrate a new, evolution-based way of using computers and to recognize its potential almost 30 years ago:

> Computer technology is now entering a new phase, one in which it will no longer be necessary to specify exactly how the problem is to be solved. Instead, it will only be necessary to provide an exact statement of the problem in terms of a "goal" and the "allowable expenditure," in order to allow the evolution of a best program by the available computation facility. . . . The old saw "the computer never knows more than the programmer" is simply no longer true.[12]

Another pioneer in the application of evolutionary ideas to computing was John Holland of the University of Michigan. During the 1940s he was involved in the development of IBM's first commercial electronic calculator, the 701, and was intrigued by the problem of getting computers to learn in a bottom-up fashion from their environment. After completing his doctoral dissertation at the Massachusetts Institute of Technology in the late 1950s on the design of a parallel-processing computer (and thereby earning the first American Ph.D. in computer science), he discovered R. A. Fisher's landmark work on biological evolution published in 1930, entitled *The Genetical Theory of Natural Selection*. This book was the first attempt to provide a mathematical account of evolutionary theory, and in it Holland saw the potential of using a form of artificial evolution to enable computers to learn, adapt, and develop intelligence on their own.

Holland's intimate knowledge of computers and his doctoral research on the design of a multiprocessor computer led him beyond the asexual mutations and parent-offspring comparisons used by Fogel's group. Holland realized that in biological evolution many processes were occurring simultaneously. Natural evolution works using large populations of organisms. Each organism interacts with its environment, and those that are most successful in surviving and reproducing leave behind the most offspring for the next generation. In addition, most multicellular organisms use sexual reproduction in which genetic material from two parents is shuffled into new combinations involving a genetic crossover process and results in offspring that are both like and yet different from either parent. By adhering closely to these biological principles, Holland and his students developed what is now known as the genetic algorithm.

A genetic algorithm begins by generating a random population of binary strings, that is, a series of zeros and ones of a certain length such as 0100101110010. Each of these strings is analogous to an organism's genome and represents a tentative solution to the problem at hand. The problem may be as trivial as finding the value of the square of 13 (in which case the string would simply be treated as a binary number). Typically, however, it is a much more complex problem for which no solution or analytic technique is known to exist, such as finding the shortest path connecting 20 cities or maintaining a constant pressure in a natural gas pipeline network subject to varying user loads and occasional leaks. Each string is a type of shorthand for a computer program that can be run to provide a trial solution. These string-generated programs are then evaluated, with those most successful (for example, the top 10%) retained and the others eliminated. The selected strings are then allowed to pair off and "mate" with each other, each mating resulting in two offspring. For example, imagine two relatively successful parent strings 10000 and 01011. If the position between the second and third bits had been randomly chosen as the cross-over point, the two parent strings would be cut in this position, yielding 10-000 and 01-011. By combining the first part of the first parent with the second part of the second parent, one offspring would be 10011; and combining the second part of the first parent with the first part of the second parent would generate the other offspring 01000. Occasionally, a bit or two is randomly flipped to simulate mutation,[13] and each string of each new generation is again evaluated. This process is repeated again and again, selecting the best strings and allowing them to mate and reproduce until a certain generation limit or error criterion is reached.[14]

Holland was betting that the processes of cumulative blind variation and selection that over many millions of years led to remarkable instances of biological adaptation and knowledge could be exploited on the computer to evolve useful algorithms using the computer's speed to compress hundreds of generations into mere seconds. It turns out that he won his bet, ultimately finding genetic algorithms capable of quickly and reliably evolving solutions to problems of daunting complexity. Scientists and engineers the world over are now using these algorithms to solve problems in many different areas. General Electric applied them to design gas and steam turbines and to make the jet engines of the new Boeing 777 more efficient. The *Faceprints* system developed at New Mexico State University allows a witness of a crime to evolve an image of a suspect's face using a system not unlike Dawkins's blind watchmaker program described earlier. The Prediction Company in Santa Fe, New Mexico, with the support of a U.S.-based affiliate of the Swiss Bank Corporation, uses genetic algorithms to make predictions useful for currency trading. Genetic algorithms have also helped design fiberoptic telecommunication networks, detect enemy targets in infrared images, improve mining operations, and facilitate geophysical surveys for oil exploration.[15] Even personal computer spreadsheet users can now find solutions to financial problems with genetic algorithms.[16]

But the evolution of evolutionary computation techniques was not to stop there. John Koza, a former student of Holland working at Stanford University, saw limitations in representing computer programs as unidimensional strings of zeros and ones. Since most complex computer programs are organized in a hierarchical fashion with higher-order routines making use of lower-order subroutines, he looked for a way of applying cumulative blind variation and selection directly to hierarchically structured computer programs. He did this with a computer language called LISP, which is structured in such a way that it lends itself particularly well to the crossover technique of sexual reproduction.

Each LISP program is structured as a hierarchical tree composed of mathematical and logical operators and data. In the technique he named genetic programming, Koza would first generate a random population of such program trees, including the operators he believed could help solve a particular problem. Programs represented by the trees would then be run and their results evaluated. Most of these programs, since they were randomly generated, provided very poor solutions, but at least a few would always be better than others. As in the genetic algorithm technique, programs that achieved the best results would be mated, and offspring programs would be produced by swapping randomly chosen branches--one each from each of the two parent trees--to produce two new program trees that were both similar to and yet different from their parents (figure 13.1). These new program trees would be evaluated, and the process continued until a program providing a satisfactory solution was found.

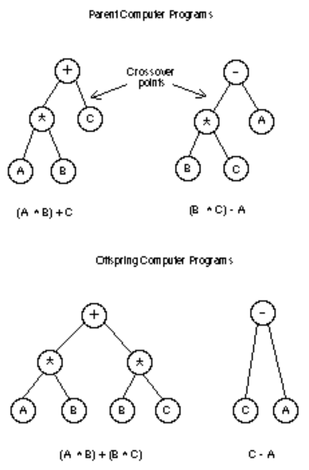

Figure 13.1 Sexual reproduction in genetic programming.

Koza's 1992 book *Genetic Programming: On the Programming of Computers by Means of Natural Selection* and companion videotape offer a veritable tour de force of genetic programming applications.[17] These include, among others, the solution of algebraic equations, image compression, robot arm control, animal foraging behavior, broom balancing, game-playing strategies, and backing up a truck to a loading dock. In applying genetic programming to the problem of finding the relationship between the distance of a planet from the sun and the time taken by the planet to complete one revolution around the sun, Johannes Kepler's third law of planetary motion was found, which relates the cube of the distance to the square of the time. Particularly intriguing was that on its way to this conclusion, the technique found a less accurate solution to this relationship that was the same as the one Kepler first published in 1608, 10 years before he discovered his error and published the correct version of his third law.[18] We can only wonder whether Kepler would have come up with the correct version years earlier if he had had access to genetic programming and let the program run beyond the intial solution.

In offering these many examples, Koza demonstrated that valuable computer programs can be developed by generating a random population of programs and then using a technique modeled directly after natural selection that takes the best programs and allows them to create new progeny sexually. Whereas such a technique would be hopelessly slow and ineffective if done manually, the recruitment of high-speed computers for this task has made it not only feasible but highly practical. Indeed, Koza showed that useful programs can be developed in this manner after as few as 19 generations. Among them are those that can generate crawling and walking behavior in simulated insects, perform aspects of natural language processing, make optimal bids and offers in commodity trading, perform financial analyses, control robots, generate art, and even produce bebop jazz melodies.[19]

Evolutionary computing techniques may turn out to be one of the most important developments in computer science of the second half of the twentieth century. The techniques are still quite young, but they are already beginning to find important commercial applications, and a number of regular scientific meetings, new journals, and electronic forums are available for disseminating the findings of research in this area.[20]

But the evolutionary approach to computing still encounters much resistance. The idea of using populations of randomly generated digital strings or LISP trees with trial-and-error elimination to write a computer program is so radically different from traditional approaches to software development that such resistance is perhaps not surprising. As science writer Steven Levy recounts from an interview with Koza:

> Many traditional programmers, Koza explains, deplored the fact that a degree of chance was involved, both in the initial randomization and the probabilistic choices of strings that survived to the next generation. "Some people just broil at the idea of any algorithm working that way," says Koza. "And then you've got complaints that its sloppy. Because the first thing that comes to your mind when you hear about crossover is, `Oh, I can think of a case where crossover won't work.' Of course you can think of a case. The key to genetic algorithms [as well as natural selection] is the population, that thousands of things are being tried at once, most of which don't work out--but the idea is to find a few that do."[21]

Indeed, many of the criticisms are not unlike the arguments marshalled against biological evolution itself. How can a complex, adapted system, be it a living organism or a computer program, possibly emerge from chance and randomness? How can such a wasteful procedure, in which almost all of the blindly generated entities in the initial and subsequent populations are eliminated, be effective in solving a complex problem, be it one imposed by a researcher on a computer program or one imposed by nature on a species? The power of nonrandom cumulative selection, of course, is the answer to these questions, and it works on the computer even better (or at least a lot more quickly) than it does in nature. And so the power of selection has begun to change the role of the computer from that of a recipient of the human programmer's knowledge to that of a generator of new knowledge.

Computers as Wind Tunnels: Simulations and Virtual Reality

Captain Emerson was definitely on edge as he lined up the Boeing 747 jumbo jet on his final approach to runway 27. He had spent thousands of hours in the air as an airline pilot, but this would be his first attempt to land the world's largest commercial passenger plane. In many ways, the Boeing 747 handled much like the much smaller 737 he knew so well. But the view of the runway from the cockpit situated four stories above the landing gear was quite unlike that of any plane he had flown. Believing that he still had several meters of altitude left when in reality he did not, the jumbo jet hit the runway hard, with a lot more force than even the monstrous landing gear of the 747 could absorb. Not only would this be the captain's first landing of a 747, it would also be his first *crash* landing.

Fortunately, however, Captain Emerson's error resulted in no injuries and no damage to the aircraft. For no actual aircraft or flying was involved. Rather, the captain had been training on a $16 million full-flight simulator, a piece of equipment that in some respects is actually more complex than the aircraft whose behavior it mimics. And so Emerson was able to learn from his error with no risk to life, limb, or equipment. He might well make a few more rough landings on the simulator, but none would be as rough as the first, and most would be better than the one before. When his turn would finally come to pilot a real 747, his first landing would seem just like one more of the many he had already performed in simulation.

The widespread use of flight simulators for pilot training is just one example of how computers are being applied to provide learning experiences and research opportunities that would otherwise be prohibitively expensive, time consuming, or risky. Although a certified flight simulator can cost many millions of dollars, many inexpensive simulation-based programs, including less sophisticated versions of flight simulators, are now available for use on personal computers. For example, the popular SimCity program created for IBM-compatible and Macintosh computers allows the user to design, build, maintain, and operate a simulated city complete with buildings, transportation systems, power-generating plants, police and fire departments, and occasional disasters such as fires, earthquakes, airplane crashes, nuclear power plant meltdowns, and even monster attacks. Another program called SimLife allows the user to design a planet, populate it with certain organisms, and allow simulated evolution to take place.

But specially designed simulation software is really not necessary to exploit selection on a computer to solve problems. Computer spreadsheets allow accountants and budget planners to investigate the consequences of various financial actions. Income tax preparation software permits tax-payers quickly and easily to investigate the consequences of various deductions and filing options, such as whether a married couple should file joint or separate returns. Drawing and design programs allow architects, engineers, and artists to try out their ideas for skyscrapers, bridges, lawnmowers, coffee makers, and sculptures without lifting a T square, hammer, screwdriver, or chisel. Even the ubiquitous word processor can be seen as a simulation of a typewriter and sheet of paper. Instead of making marks on paper, the word processor allows the writer to put easily changed characters on computer displays and magnetic disks, making revising and editing easier to accomplish than working directly on paper. More sophisticated examples are the simulations of mammalian nervous systems (to be considered below) and of new computer architectures using existing computers.

Although these examples may initially appear quite diverse, they are all alike insofar as the computer is used to simulate an environment that is more conducive to repeated cycles of trial-and-error elimination than the actual environment. Computer-aided variations can be generated more easily, more quickly, and more cheaply, thereby increasing the chances and reducing the cost of finding one that provides a good fit to a problem. In this way, the computer may be seen as a type of digital wind tunnel. We saw in chapter 10 how Wilbur and Orville Wright tested many of their ideas for aircraft design using quickly produced, inexpensive models in a wind tunnel, while their French competitors proceeded directly to full working implementations of their designs. By simulating the conditions of flight using models of propellers and wing shapes, the Wright brothers solved the problem of powered flight relatively quickly, and the French went from one crash to another. As aeronautical historian Walter Vincenti noted, "use of vicarious trial, both experimental and analytical, was a strength of the Wright brothers in comparison with their French contemporaries."[22] In much the same way, the computer now provides a means by which homework assignments, hair styles, public transportation systems, and new supercomputer architectures (to name only a few examples) can be proposed, tested, and refined before producing a printed copy or real working model or prototype.

This vicarious variation and selection has much in common with and extends the use of the human brain to propose and test solutions to problems, as discussed in chapter 9. We saw there that important functions of thinking are generating, testing, and selecting thought trials. Instead of having repeatedly to rearrange all the furniture in his living room until an acceptable new arrangement is found, the customer in the piano showroom can mentally generate, evaluate, reject, and finally select a new furniture arrangement that will accommodate his new piano. But although the brain has quite good verbal, visual, and auditory pattern-generating skills, it is less adept at performing certain complex mathematical computations and logical reasoning. So by including these mathematical and logical aspects in a simulated computer environment, a quite powerful combination of human intellectual and machine computational resources is formed for finding solutions to difficult problems.

The ultimate development of this brain-computer collaboration can be seen in the creation of what is known as virtual reality. The goal of developers of virtual reality systems goes far beyond the small video displays and anemic loudspeakers found on most computers by making it possible for users to experience a simulated environment in much the same way that they experience the real world. Instead of seeing a new design for a home or office building on blueprints, architect and client can instead don display goggles, step onto a steerable treadmill, and walk through a virtual building. An ultrasound-generated image placed between physician and patient allows a surgeon to see inside the patient to guide the initial critical moves of the scalpel. Molecular engineers can now touch and manipulate atoms the size of tennis balls to create new molecular structures that may have important scientific, engineering, and medical uses. Even virtual wind tunnels have been developed that allow aerospace and automotive engineers to enter the airstream virtually with a new vehicle design to examine from several perspectives how air flows past it at different speeds.[23]

Daunting technological challenges must still be overcome to make virtual reality a useful and accessible tool,[24] but virtual environments appear to be the ultimate step in facilitating variation and selection processes for generating new knowledge to solve new problems. A virtual reality-based architect can eliminate walls and add windows with a sweep of the hand to determine whether the change should be kept or undone. The wings of a new jet fighter can be modified instantly to see if some better combination of stability and maneuverability can be achieved. And the surgeon can rule out certain interventions without having first to subject the patient to exploratory surgery. Virtual reality is still in its infancy, but it seems inevitable it will play an increasing role in furthering our knowledge and technology in many fields due to the way that such virtual environments can greatly facilitate and accelerate the generation, variation, evaluation, and selection of potential solutions to complex problems.

Neural Networks

The relatively new field of artificial intelligence involves the creation of computer hardware and software designed to mimic and in some ways to surpass the perceptual, thinking, and reasoning powers of the human brain. Although many observers of this field have been disappointed with the results to date, there has been considerable progress in work that involves computer modeling and simulation of the workings of the brain and nervous system. Based on our current understanding of the brain as consisting of interconnected networks of billions of relatively simple units (neurons), developers of neural networks create computer simulations of neurons (sometimes referred to as neurodes) and connect them up in various ways to form networks that can learn and act intelligently in some way.[25] Such networks are able to recognize human speech and written letters, play games such as backgammon, and determine whether sonar echoes have been reflected from undersea rocks or mines.[26]

One of the most widely used neural network architectures is known as the backpropagation network (figure 13.2). This is typically a three-layer network with a set of input neurodes connected through a set of middle-level neurodes to a set of output neurodes. It can produce certain desired responses (output patterns) when presented with certain input patterns. For example, the input patterns might correspond to received sonar reflections of the type used by submarine operators, and the desired output would be a pattern indicating the presence of either a rock or a mine, providing in effect an automatic and accurate classification of the object being sensed.

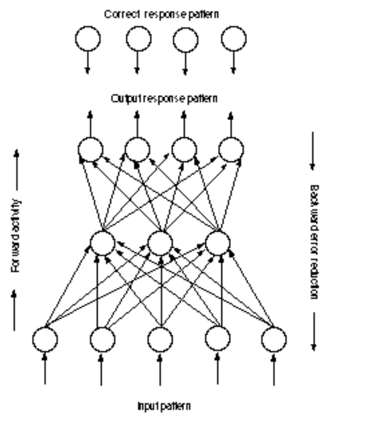

Figure 13.2 Back propagation neural network (after Caudill & Butler, 1992).

For a neural network to behave in this way, a particular pattern of connection strengths (analogous to the strengths of synaptic connections between neurons) must first be found between each neurode and the neurodes in neighboring levels. With the connection strengths among the neurodes initially set randomly, a pattern is presented to the input level that causes the network to produce an output pattern that will be interpreted as either "rock" or "mine." Since the initial connection weights are set at random, the initial outputs are most likely to be wrong. However, a training procedure is used whereby each output pattern is compared with the desired (correct) output pattern, and changes to the connection strengths are made according to a mathematical formula that changes the connection strengths of those neurodes that most contributed to the error. This use of error to go backward into the network to change the connection weights gives the backpropagation network its name. After many iterations of this procedure--providing an input, observing the output, noting the discrepancy between the actual output and desired output, and changing the synaptic weights among the neurodes to reduce the error--the network may learn to classify correctly not only input patterns on which it was trained, but also new ones it has not experienced before. In addition to classifying sonar signals, such networks can remove noise from medical instrument readings, produce synthesized speech, recognize written characters, and play various board games.[27]

It is of particular interest to note that no trial-and-error and no blind variation and selection are involved in the actual training and functioning of a backpropagation neural network. Instead, during training the difference between the network's actual output and the desired output is used to modify synaptic weights in a deterministic manner calculated to reduce error the next time that input pattern is encountered. This is the first clear example we have seen in this book of adaptive change resulting from an instructionist process--the instruction provided by that part of the computer program that trains the network.

But this instructionist approach to training a neural network has certain noteworthy limitations. First, the designer must initially decide on the number and organization of the neurodes in the network. Particularly important is the number of middle-level neurodes--too few, and the network will not be able to distinguish between subtle but important differences in input patterns; too many, and the system may not be able to generalize what it has learned to new input patterns on which it was not trained. And since there is no way to know beforehand the optimum number of such units, this initial design is actually the result of trial and error on the part of the network developer.

Second, the correct corresponding output, that is, the proper classification for each encountered input pattern, *must already be known* to train the network. A sonar-detecting backpropagation network cannot be used to distinguish rocks from mines if it is not already known which signals indicate rocks and which indicate mines. In other words, the instructionist learning procedure used by backpropagation neural networks can work only if the instructor already knows all the right answers for the training inputs. So this process cannot be used to discover new knowledge or develop new skills, but is a way of transferring knowledge from one source to another.

Third, backpropagation neural networks cannot always be trusted to find just the right combination of connection weights to make correct classifications because of the possibility of being trapped in local minima. The instruction procedure is one in which error is continually reduced, like walking down a hill, but there is no guarantee that continuous walking downhill will take you to the bottom of the hill. Instead, you may find yourself in a valley or small depression part way down the hill that requires you to climb up again before you can complete your descent. Similarly, if a backpropagation neural network finds itself in a local minimum of error, it may be unable to climb out and find an adequate solution that minimizes the errors of its classifications. In such cases, one may have to start over again with a new set of initial random connection weights, making this a selectionist procedure of blind variation and selection. One might also use other procedures such as adding "momentum" to the learning procedure, analogous to the way a skier can ski uphill for a while after having attained enough speed first going downhill, or adding noise to the procedure to escape such traps, again a form of variation and selection.

Finally, for those looking to research on neural networks to further our understanding of how real neurons work in real brains, backpropagation neural networks are not biologically plausible. Unlike the neurodes and their interconnections in a backpropagation network that conduct signals in both directions (one way for responding, the other direction for learning), biological neural pathways conduct signals in one direction only. Also, much if not all human learning occurs without the direct instruction and the patient training of the type backpropagation networks require. So although these networks can be useful in quite a number of interesting applications, their instructionist nature does impose limitations on both their adaptive flexibility and their applicability to biological neural systems.[28]

Given the limitations of instructionst training, it should not be surprising that selectionist approaches to designing neural networks have also been developed.[29] Some of the first were developed by Andrew Barto and his colleagues at the University of Massachusetts in the early 1980s. These researchers demonstrated that neural networks could learn to solve difficult problems by a selectionist process referred to as reinforcement learning as opposed to the supervised learning characteristic of backpropagation and other instructionist networks.[30] For example, one system learned to balance a pole hinged to a movable cart,[31] and others solved spatial learning problems.[32]

Another type of selectionist neural network is known as a competitive filter network. Like the typical backpropagation network, it consists of three groups of interconnected neurodes--the input layer, the middle or competitive layer, and the output layer. The network learns to make useful output responses to input patterns, but without the help of an instructor. An example is learning to recognize spoken words. For such a task the input layer encodes the sounds of the detected word and sends the resulting pattern of activity on to the middle layer. Since each neurode in the middle layer is connected to each and every neurode in the input layer, every middle-layer neurode is activated to some extent by the input pattern. But each middle-layer neurode has a different pattern of (initially random) weights that mediates the effect of the input pattern. Those having the pattern that best matches the input pattern are the most active.

In addition, connections among the middle-layer neurodes are arranged so that a highly active neurode excites neighboring neurodes while inhibiting those neurodes farther away. The net result of all this simulated neural activity is *competition* among the middle-layer neurodes, with the best-fitting ones sending their signals on to the output layer and allowed to adjust their weights so that they match the input signal even more closely than they did previously. After many trials, such a network can learn to categorize input patterns into meaningful and useful categories so that one and only one middle-layer neurode will respond to all or nearly all pronunciations of a specific word. Teuvo Kohonen, who did extensive work with such competitive filter networks, used such a system to create one of the first voice typewriters.[33]

It is not difficult to see the evolutionary processes of variation, selection based on competition, and retention operating in such a network on the connection weights between neurodes. The initial distribution of weights for the middle-layer neurodes is usually determined randomly. Then competition occurs among these neurodes, with the most active one winning and selected to be modified to match the original input pattern even better. But because this modification is directed toward a better matching of the just-received input pattern, there is no guarantee that this modification of weights will be retained, as more input patterns are received and further rounds of activation, competition, selection, and modification of connection weights take place.

Another class of neural networks known as adaptive resonance networks also use processes of variation, competition, and selection. For them,

> . . . the basic mode of operation is one of hypothesis testing. The input pattern is passed to the upper layer, which attempts to recognize it. The upper layer makes a guess about the category this bottom-up pattern belongs in and sends it, in the guise of the top-down pattern, to the lower layer. The result is then compared to the original pattern; if the guess is correct . . . the bottom-up trial pattern and the top-down guess mutually reinforce each other and all is well. If the guess is incorrect . . . the upper layer will make another guess. . . . thus, the upper layer forms a "hypothesis" of the correct category for each input pattern; this hypothesis is then tested by sending it back down to the lower layer to see if a correct match has been made. A good match results in a validated hypothesis; a poor match results in a new hypothesis.[34]

Gerald Edelman, whose neural Darwinism was discussed in chapter 5, has also been actively involved with his associates in applying Darwinian selection to neural networks to simulate the adaptive functioning of the brain.[35] Using what he calls synthetic neural modeling, he and his associates have created a series of neural simulations named after Darwin that explore and demonstrate the principles of neuronal selection. Darwin III is a computer simulation of a sessile (seated) creature possessing a head with a movable seeing eye and a jointed arm that has both touch and kinesthetic sensors.[36] Inhabiting a computer world of simulated stationary and moving objects of various shapes and textures, Darwin III learns to grasp certain objects and repel others. Although certain initial biases are built into Darwin III, for example, a preference to grasp round, smooth objects and reject square, bumpy ones, the actual perceptual categories and behaviors are not programmed. Rather, Darwin III learns to fixate visually and track objects, distinguish different types of objects, and grasp some while rejecting others through Darwinian selection of connection weights between the neurodes, in much the same way it appears that the human brain is able to learn through the variation and selection of synaptic connections among its neurons. With the recent creation of Darwin IV, Edelman and his associates have moved beyond computer simulations and created a working robot that performs the tasks of its predecessors in a real-world environment of physical objects.[37]

Before leaving our discussion of neural networks, let us return briefly to the instructionist networks described at the beginning of this section. It will be recalled that these systems of simulated neurons can indeed grow in adapted complexity through an instructionist process by which synaptic weights are modified in the proper directions without variation and selection. This being the case, however, we must wonder how it is that these network architectures and training procedures came to be, being themselves examples of adapted complexity and thereby posing additional puzzles of fit. Unless they were discovered in some novel manner completely unlike that responsible for other scientific and technological innovations, we must suspect that they owe their design to repeated cycles of blind variation and selection, not in the working of the networks themselves but in repeated overt and vicarious (cognitive) generation of the networks and their subsequent testing by scientists. And since, as we saw earlier in this chapter, computers can now simulate such selection processes, we should not be too surprised to learn that recent work in genetic programming has shown that the computer itself can be used to evolve neural networks to solve problems and perform various tasks.[38]

It can be argued that the digital computer is the most important technological tool of the twentieth century, but it is quite often viewed as a sophisticated electronic combination of file cabinet, calculator, and typewriter. Most computers are indeed still used in these ways. But we are now beginning to see exciting new applications that go far beyond their standard office and scientific functions. Computers have made it possible to model the evolutionary process itself and thereby convincingly demonstrate the cumulative power of the combined processes of blind variation and selective retention. They can be used to simulate real-world environments, allowing scientists and engineers to test many trial solutions rapidly, economically, and safely. They are being programmed to invent, test, and improve their own solutions to exceedingly complex problems using the techniques of genetic algorithms and genetic programming.

One cannot help but wonder where all this will ultimately lead. Will future computers and programs and perhaps even robots be bred in ways analogous to the ways that chickens, ornamental flowers, and corn are bred today, only much faster? Will programs and the machines they run on begin to evolve intelligence and understanding that will eventually approach and perhaps even surpass that of their human designers? Or will certain undiscovered limits to the evolutionary potential of computer hardware and software prevent them from achieving anything near the adapted complexity that organic evolution has achieved over the last four billion years? These are questions that only time can answer. But it now seems almost a certainty that as computers themselves continue their own technological evolution, thereby acquiring ever-increasing memory, processing power, speed, data storage, and audiovisual capabilities, the evolutionary processes first discovered among carbon-based life forms will be used increasingly within silicon-based machines.

[1]Goldberg (1986; quoted in Levy, 1992, p. 153).

[2]But the effects of evolution can be seen if one carefully studies the right species in the right location for a long enough time, as in the Grants' 20-year study of the evolution of Darwin's finches on the Galápagos Islands (see Weiner, 1994).

[3]Dawkins (1986).

[4]A Macintosh version of Dawkins's program is available from the publisher W. W. Norton in New York.

[5]Dawkins (1986, pp. 59-60).

[6]Much of the following account of Ray's work is based on Levy (1992, pp. 216-230).

[7]Quoted in Levy (1992, p. 221).

[8] See Dawkins (1986, chapter 7).

[9] Readers with access to either an IBM-compatible computer or a Unix workstation or mainframe computer can also observe the processes of evolution using programs available by anonymous FTP at tierra.slhs.udel.edu or life.slhs.udel.edu.

[10]See Levy (1992) for a more detailed account of the work of pioneers in the field of artificial life.

[11]Bell (quoted in Levy, 1992, p. 320).

[12]Fogel, Owens, & Walsh (1966, p. 113).

[13]One particularly surprising outcome of the work done by Holland and his students on genetic algorithms was the importance of crossover and the relative insignificance of mutation in the evolutionary process. It was found that the new recombinations provided by the crossover of genes in sexual reproduction allowed the genetic algorithm to find building blocks during the evolutionary process so that fitness could better be accumulated from one generation to the next. In contrast, mutation (the occasional random changing of a 0 to 1 or vice versa) turned out to be relatively unimportant, serving primarily as a way of resurrecting old algorithms that had already been discarded but whose descendants would nonetheless provide useful solutions.

[14]A nontechnical account of genetic algorithms can be found in Holland (1992). A more comprehensive, technical treatment is provided by Goldberg (1989).

[15]See Goldberg (1994) for more information on these and other real-world applications of genetic algorithms.

[16]Evolver, developed by Axcellis in Seattle, is designed to work in conjunction with Microsoft Excel.

[17]Koza's follow-up book *Genetic Programming II*, was published in 1994 together with another accompanying videotape.

[18]Koza (1992, pp. 257-258).

[19]For more information on these and other applications of genetic programming, see Kinnear (1994) and appendix F of Koza (1994). Singleton (1994) provides information on how genetic programming can be done with the widely used C++ programming language.

[20] The International Conference on Genetic Algorithms had its first meeting in 1985 and has met on every odd-numbered year since. The Workshop on the Foundations of Genetic Algorithms meets in even-numbered years. Other regular meetings on related topics include Parallel Problem Solving from Nature, the IEEE International Conference on Evolutionary Computing, Artificial Life Workshops, and the European Conference on Artificial Life.

Three young journals that publish related research are *Artificial Life*, *Adaptive Behavior*, and *Evolutionary Computation*, all published by the MIT Press. Readers with access to the Internet can learn more about genetic algorithms by subscribing (for free) to the genetic algorithm (ga-list-request@aic.nrl.navy.mil) and genetic programming (genetic-programming-request@cs.stanford.edu) electronic mailing lists. A repository of programs and papers on genetic programming is available on the Internet through anonymous FTP from the site ftp.cc.utexas.edu in the pub/genetic-programming directory.

[21]Levy (1992, p. 179; bracketed material in original).

[22]Vincenti (1990, p. 248).

[23]See Larijani (1994) for an introduction to virtual reality and many additional examples of its use and potential.

[24]See Gibbs (1994).

[25]The field of neural networks overlaps with the study of parallel distributed processes (PDP), both of which are referred to as connectionist approaches to artificial intelligence.

[26]See Caudill & Butler (1990) for a nonmathematical introduction to neural networks (including the ones described here) and what they can be made to do.

[27]Caudill & Butler (1990, pp. 194-195).

[28]Two other neural network architectures that learn by instruction are known are perceptrons and adalines.

[29]The term stochastic is often applied to selectionist neural networks, referring to the random variations used to develop the networks.

[30]Barto (1994, p. 202) uses the term supervised learning to refer to tasks where a teacher or supervisor not only knows the right answer but also how the network should change its internal organization in order to reduce its error. This is therefore an instructionist approach to learning. In contrast, reinforcement learning employs a critic who provides the system information about the degree of error only and "does not itself indicate how the learning system should change its behavior to improve performance; it is not directed information. . . . [The system] has to probe the environment--perform some form of exploration--to obtain information about how to change its behavior" (pp. 203, 218). Reinforcement learning is selectionist in that the learning system must be active in exploring possible solutions using the information provided by the critic to reject some but select and cumulatively modify others.

[31]Barto, Sutton, & Brouwer (1981).

[32]Barto & Sutton (1981).

[33]See Caudill & Butler (1990, pp. 105-109).

[34]Caudill & Butler (1990, p. 208).

[35]Edelman (1993); Friston et al. (1994); Sporns & Edelman (1993).

[36]See Edelman (1989, pp. 58-63; 1992, pp. 91-94); Reeke & Sporns (1990). Kinesthesia refers to sensory information on the position of body parts; for example, lets you know where your hand is when you are not looking at it.

[37]Edelman et al. (1992). See also Levy (1994) for a brief description of Darwin IV as well as an interesting account of Edelman's work, personality, and ambitions.

[38]See Albrecht, Reeves, & Steele (1993); Koza (1992, pp. 513-525). Another computational technique used to solve problems like those encountered in neural networks is referred to as simulated annealing. This is a process by which a solution is found to a complex problem (such as determining the structure of a molecule based on data provided by magnetic resonance imaging) by taking a tentative solution and effectively shaking it up to see if a better solution can be found. Since this shaking introduces a source of blind variation in an attempt to find and select a better solution, the evolutionary parallel is quite obvious. See Davis (1987) for a collection of chapters providing an introduction to simulated annealing and its applications.