cover story

TAKING A PAGE FROM DARWIN'S *ON THE ORIGIN OF THE SPECIES*, COMPUTER SCIENTISTS HAVE FOUND WAYS TO EVOLVE SOLUTIONS TO COMPLEX PROBLEMS

What is evolutionary computation?

DAVID B. FOGEL
Natural Selection Inc.

THE PRINCIPLE OF EVOLUTION is the primary unifying concept of biology, linking every organism together in a historical chain of events. Every creature in the chain is the product of a series of "accidents" that have been sorted out thoroughly under selective pressure from the environment. Over many generations, random variation and natural selection shape the behaviors of individuals and species to fit the demands of their surroundings.

This fit can be quite extraordinary and compelling [Fig. 1], a clear indication that evolution is creative. While evolution has no intrinsic purpose—it is merely the effect of physical laws acting on and within populations and species—it is capable of engineering solutions to the problems of survival that are unique to each individual's circumstance and, by any measure, quite ingenious.

Imagine what harnessing the evolutionary process within a computer might do. It could provide a means for addressing complex engineering problems—ones involving chaotic disturbances, randomness, and complex nonlinear dynamics—that our traditional algorithms have been unable to conquer. Indeed, the field of evolutionary computation is one of the fastest growing areas of computer science and engineering for just this reason; it is addressing many problems that were previously beyond reach, such as rapid design of medicines, flexible solutions to supply-chain management problems, and rapid analysis of battlefield tactics for defense. Potentially, the field may fulfill the dream of artificial intelligence: a computer that can learn on its own and become an expert in any chosen area.

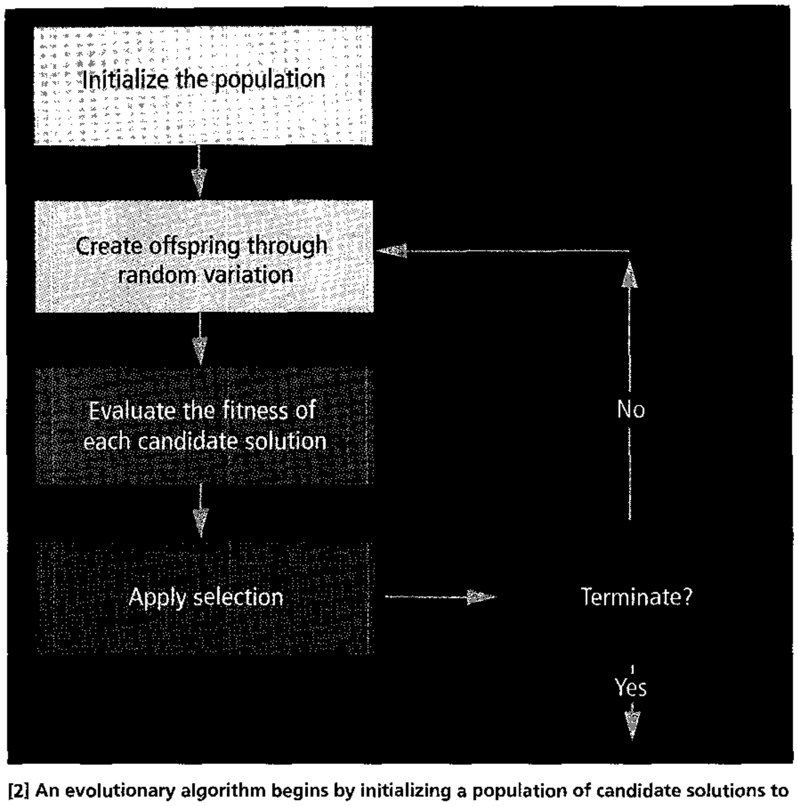
In the most general terms, evolution can be described as a two-step iterative process, consisting of random variation followed by selection. The link between this description of evolution and the optimizing algorithms that are the hallmark of evolutionary computation is conceptually simple.

Just as natural evolution starts from an initial population of creatures, the algorithmic approach begins by selecting an initial set of contending solutions for a particular problem. The set may be chosen by generating solutions randomly or by utilizing any available knowledge about the problem.

These "parent" solutions then generate "offspring" by a preselected means of random variation. The resultant solutions are evaluated for their effectiveness—their "fitness"—and undergo selection. Just as nature imposes the rule of "survival of the fittest," those solutions that are the least fit are removed from further consideration, and the process is repeated over successive generations [Fig. 2].

[1] Evolution copes with problems in ways that might never occur to people, who tend to think linearly. Witness the leafy sea dragon, whose limbs and back take on the form of the surrounding vegetation, concealing it from predators.

(Source: *A Natural Selection: Marvels and Oddities of the Natural World*, R. Morecroft, Simon and Schuster, 1993)


Initialize the population
Create offspring through random variation
Evaluate the fitness of each candidate solution
Apply selection
Terminate?
No
Yes
[2] An evolutionary algorithm begins by initializing a population of candidate solutions to a problem. New solutions are then created by randomly varying those of the initial population. All solutions are measured with respect to how well they address the task. Finally, a selection criterion is applied to weed out those that are below par. The process is iterated using the selected set of solutions until a specific criterion is met.

THE ASSUMPTION PROBLEM

To be useful, traditional algorithms for discovering the most appropriate solutions—optimization algorithms—require that their users make many assumptions about how to evaluate the fitness of a solution. (These traditional means of evaluation go by many names: the fitness or cost function, the response surface, or, in engineering, the performance index.) For example, the so-called linear programming algorithms demand that the cost functions also be linear—a sum of weighted individual cost terms. Another traditional approach—gradient-based search, in which the point of zero gradient, hopefully, the maxima or minima, is sought—requires a smooth, differentiable cost function; it is unable to deal with sudden, discontinuous change.

But evolutionary algorithms require no such assumptions. Fundamentally, the performance index need only be able to rank two competing solutions; that is, it must determine that one solution is, in some way, better than another. This makes a broad range of problems that are outside the range of conventional engineering amenable to the evolutionary approach. In short, evolutionary algorithms can often resolve problems that do not yield to common numerical techniques.

In the real world, an evolutionary approach to solving engineering problems offers considerable advantages. One such advantage is adaptability to changing situations. For instance, suppose a manager must find the best schedule for operating a factory. Doubtless, there will be many constraints: availability of personnel, the number of machines, the time required to change machine settings, and so forth. Even if the manager could find an optimum schedule that would be most profitable, he or she would still need to consider that machines may break down and personnel may not arrive at work on time. In daily life, the problem faced at any moment may have diverged significantly from the problem anticipated originally.

Unfortunately, in many traditional optimization procedures, the calculation must be restarted from the beginning if any variable in the problem changes. This is computationally expensive. With an evolutionary algorithm, on the other hand, the current population serves as a reservoir of stored knowledge that can be applied on the fly to a dynamic environment. Restarting from scratch is not necessary.

Another advantage of an evolutionary approach to problem solving comes in being able to generate good enough solutions quickly enough for them to be of use. This ability is perhaps best illustrated by the classic traveling salesman problem, to wit, "Suppose a salesman must visit clients in different cities, and then return home. What is the shortest tour through those cities, visiting each one once and only once?"

This problem is simple to state...but difficult to solve. It belongs to the class of problems referred to as NP-hard, where NP stands for nondeterministic polynomial. For such problems, no known algorithms are able to generate the best answer in an amount of time that grows only as a polynomial function of the number of elements in the problem.

In fact, the number of possible tours in any traveling salesman problem increases as a factorial function of the number of cities. So for 100 cities, there are over 10^{155} different possible paths through all the cities one could try. Considering that there are only an estimated 10^{18} seconds in the history of the universe, simply applying brute force to search all possible solutions to a traveling salesman problem of even modest size is doomed to fail.

Instead, consider an evolutionary approach to discovering a useful solution to such a problem. The approach has four prerequisite steps:

- Choosing the solution representation.
- Devising a random variation operator.
- Determining a rule for solution survival.
- Initializing the population.

Once these are taken, the evolutionary algorithm will start to generate solutions, and will continue to generate better ones as time passes. Each of these steps is explained in greater detail in the following paragraphs.

EVOLUTIONARY STEPS

To represent any possible solution within the confines of a computer, a structure must be defined for the data that will encode every possible solution that it might be desirable to evaluate. Here, there is no single best choice for the representation (this is provable mathematically) so a little ingenuity is called for.

One potential data representation for the traveling salesman problem is to identify each different possibility in a permutation. To simplify, if there were six cities (including the salesman's home base), then one possible solution might be [1 2 3 4 5 6], which would indicate an order of progression. (Note that, as this problem requires a round trip—that is, a closed loop—the first element in the series is also the last. Also note that, since this is a loop, it does not matter which city the salesman actually starts from.) Any permutation of these cities would be another more or less desirable solution to the problem.

Suppose that this representation is chosen. Then the cost function, that is, the means to evaluate any candidate solution, must also be determined. Here the task is straightforward: travel as short a distance as possible. So the "cost" of any solution can be made equal to the distance of the tour, with shorter tours being favored over longer ones. Of course, things could be made more complicated, incorporating additional aspects of real-world problems such as minimizing the traveling required during peak-traffic hours or requiring that certain customers be visited, say, in the afternoon or only after other customers are visited. If so, evaluating candidate solutions might become much more complex. Complexity only makes the application of an evolutionary algorithm more pertinent, because it rapidly removes the problem from the domain of most traditional optimization techniques. For the sake of this example, let the simple total distance of the tour be the measure of the quality of a solution.

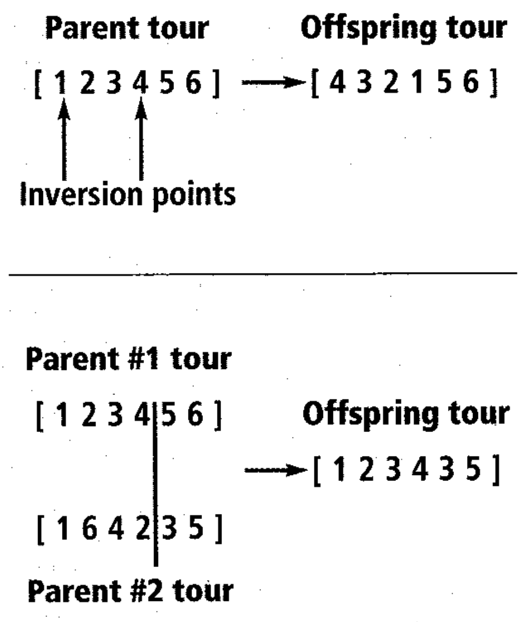
The second step is to devise a random variation operator (or operators) that can be used to generate offspring solutions from parent solutions. Many options exist. In nature, there are two general forms of reproduction: sexual and asexual. In sexual reproduction, two parents within a species exchange genetic material that is recombined to form an offspring. Asexual reproduction is essentially cloning, but mutations of various forms can creep into the genetic information passed along from parent to offspring. These operators are worth modeling in an evolutionary algorithm.

Thinking even more broadly, why not devise other variation schemes not found in nature? Examples include recombining genetic material from three or more parents and allowing a democratic vote between the parents involved in reproduction. There is virtually no limit to the types of variation operators that can be devised, nor any reason to be constrained by nature for inspiration.

The ultimate success of an evolutionary algorithm depends strongly on how well the variation operator(s), the representation, and the evaluation function are matched. Different operators will vary in usefulness with the situation. Just as with the representation, there is no single best variation operator for all problems (and this, too, is provable mathematically).

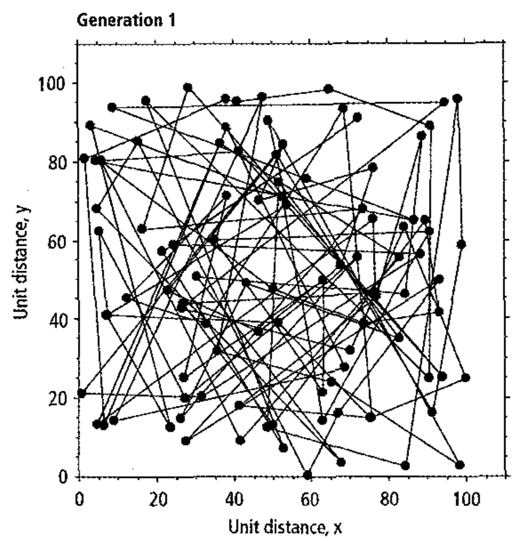
Consider two possibilities for the traveling salesman problem represented using permutations. One choice is to use a single parent and generate an offspring by randomly choosing two positions along the parent and inverting the list of cities in that segment [Fig. 3]. Another is to use two parents, choose a random point along the permutation, then take the first segment of cities from the first parent and the subsequent segment of cities from the second parent.

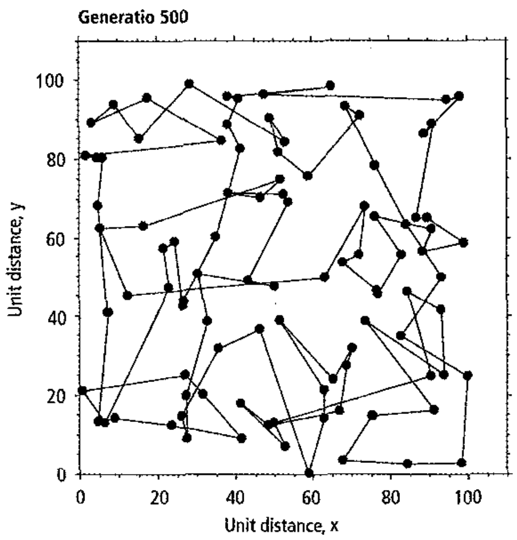
The first operator looks a bit like asexual reproduction, while the second looks something like sexual reproduction. But whereas the first variation will always generate a legal tour (each city visited once and only once), the second variation might generate illegal tours, because some offspring may contain more than one copy of some cities and no copies of others. That does not rule out using sexual reproduction; it merely dictates the inclusion of additional operations to "repair" such solutions so that they can be fixed before they are evaluated. The remedy might be to locate any city that is rep-



[3] The traveling salesman problem is: find the shortest route for visiting all the cities selected. With six cities [1 2 3 4 5 6], a single-parent operator could create an offspring solution by inverting the visiting order between two randomly chosen points [top].

With a two-parent operator, two tours could be split at some random point and the pieces joined to create an offspring [bottom]. The tour here, though, is incomplete and additional rules are needed to repair it or to prevent it from occuring.





[4] A simple iterative evolutionary algorithm was used to solve a 100-city traveling salesman problem. At the start, the researcher selected 100 possible solutions, each of which was used to generate one offspring by means of inversion [Fig. 3], yielding a total of 200 solutions. The best 100 (in terms of shortest route) were selected, with the top solution from this first generation shown at far left. The best solutions after 500, 1000, and 4000 iterations indicate the progress made as the process was repeated.

resented more than once in one solution, then replace that city with one not represented at all. Once this is done for each duplicated city, the offspring will have been repaired and become a viable contending solution. Many variation operators could be considered for the traveling salesman problem but, for illustration, the inversion operator described above will be used here.

The third step is determining a rule for selecting which solutions will survive to become parents of the next generation. As with variation, many forms of selection can be considered. One simple rule is survival of the fittest: only the handful of very best solutions in the population is retained while all the others are killed off, so to speak. An alternative is to use a sort of tournament approach, where randomly paired solutions compete for survival. Just as in professional sports, where weaker players or teams sometimes win because they get a lucky draw in the tournament, weaker solutions in a population sometimes survive a few generations under this format. This can be a plus in complex problems, where it may be easier to find new improved solutions by making variations of weaker ones than to do so by relying only on the very best. The possibilities abound, but any rule that generally favors better solutions over weaker solutions for survival is reasonable. For simplicity, only the most basic approach—survival of the fittest—will be considered here.

GENESIS

The final step is selecting the initial population. If nothing is known about how to solve the problem, then solutions can be chosen completely at random from the space of all possible solutions. In the case of the traveling salesman problem here, that would mean randomly generating a number of permutations of integers, each permutation representing one possible solution.

Alternatively, there may be some hints at good solutions available—perhaps because some other algorithm or prior knowledge can be used to generate some reasonable head start—which can be incorporated in the initial population. If those solutions prove worthwhile, they will survive and produce new variants; if they are false leads, then they will perish along with other, weaker solutions. For the example here, suppose the initial population is chosen completely at random.

A typical run of an evolutionary algorithm on a 100-city traveling salesman problem, in which the cities have been distributed at random but in a uniform manner, has the results shown in Fig. 4. The improvement at successive stages in the evolution of the tours is evident. (The program for this example, written in Matlab, is available at www.natural-selection.com under the publications section.)

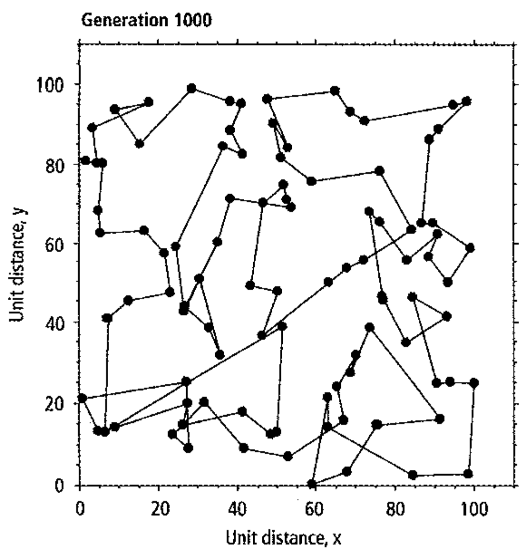
The evolutionary algorithm has searched the space of possible tours and has discovered a very good one. While it is probably not the perfect answer, it is of high quality. In all, only 400 000—or one out of every 10^{150}—possible solutions were examined, an infinitesimally small fraction of the whole. This is in line with the operating credo of evolutionary practitioners: "Solutions should be good enough and generated fast enough to be useful."

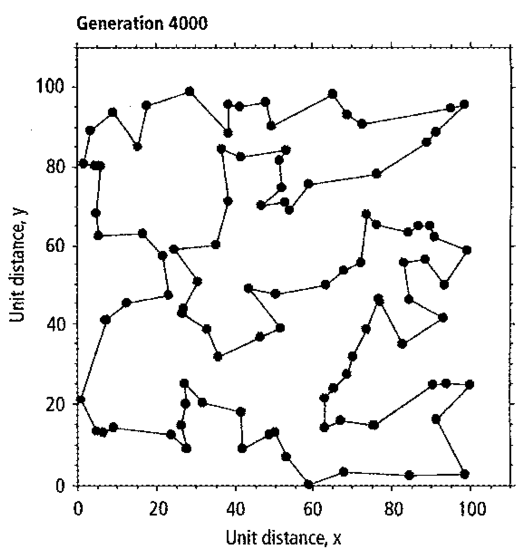
Granted, there are now many better algorithms for solving the traveling salesman problem than the above evolutionary procedure, since mathematicians have already spent much time and effort trying to solve it. But all these other techniques rely on some specific knowledge about the problem to improve their performance. They sacrifice generality in order to gain performance. Just like Deep Blue, the world champion chess program, they shine in their narrow domain of application, but do poorly outside that domain. Imagine Deep Blue playing a game of checkers: it couldn't make even the first move.

The key point is that, while it is possible to incorporate any problem-specific knowledge available and thereby take advantage of it when using evolutionary algorithms, it is not inherently necessary to the application of the techniques. This is why evolutionary algorithms can tackle an enormously broad range of problems.

IN THE FIELD

Evolutionary algorithms are already being used to solve a wide variety of real-world problems that pose significant challenges. One such problem involves the discovery of new drugs. In the case of docking a small molecule of a potential drug (called a *ligand*, meaning "something that has to be bound") into a target protein's binding site, the result depends in large part on the three-dimensional shapes of both elements. It is analogous to a lock and key: just as only the right-shaped key will open a particular lock, so only an appropriately shaped ligand will be able to bind to the target protein.





Finding that ligand is extremely difficult. Not only are there tens of thousands of candidates, but each one has many bonds that can be rotated into many different positions. Essentially, every ligand has infinitely many shapes. It is certainly true that many of these possible conformations are extremely unlikely owing to the physical and chemical properties of the molecules. Nevertheless, the number of possible profiles that each ligand can present is often enormous. This makes predicting the best conformation of a ligand as it binds to a target protein quite challenging.

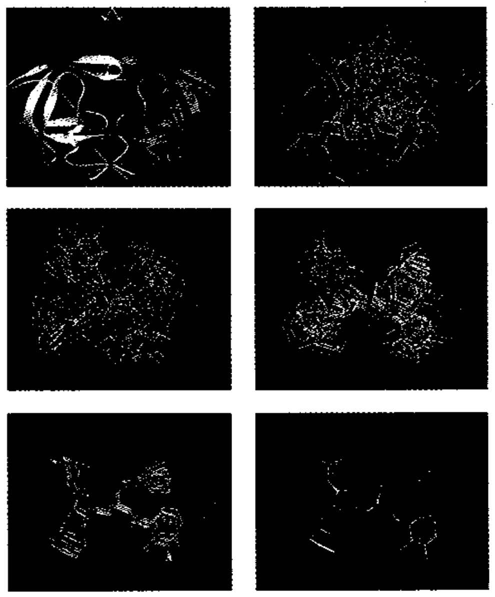
In a study published in *Chemistry & Biology*, researchers at Agouron Pharmaceuticals Inc. indicated how an evolutionary algorithm could be used for predicting the manner in which a ligand would dock into HIV-1 protease as a potential drug against AIDS. The task was to find the most energetically favorable conformation for the ligand when it attached to the binding site of the protein. Several factors were believed to be important, including:

- The steric (three-dimensional) fit of the ligand, based on how well it complements the surface structure of the binding site on the protein.
- Electrostatic interactions between atoms.
- Van der Waal's forces, which dictate essentially that two atoms cannot be in one and the same place.

In light of these criteria, any potential conformation of a ligand can be scored in terms of how well it binds to the target protein.

An evolutionary algorithm was devised to operate on a population of over 1000 different potential conformations of a candidate ligand, with variation randomly altering the rotation angles of the ligand, and selection sorting out which shapes appeared to be more energetically favorable than others. Figure 5 shows a successful evolution of the ligand, named AG-1343, docking into HIV-1 protease after completion of the evolutionary algorithm and after post-processing by a gradient minimization on each rotation angle. Comparing it to the experimentally determined crystal structure indicates that there is very little discrepancy between the predicted conformation derived from the evolutionary algorithm and the actual observation gleaned from nature. In summary, the experiments give independent verification that the evolutionary algorithm did indeed find the correct answer.

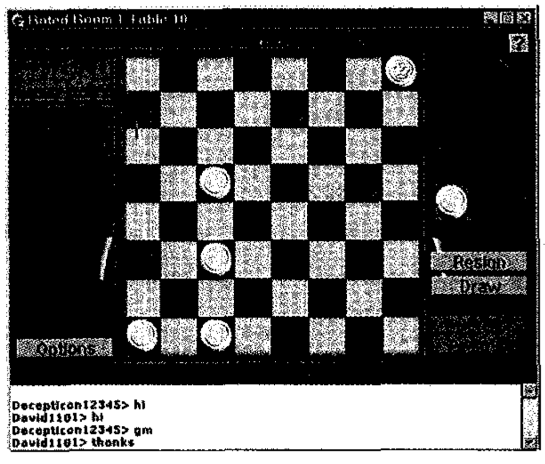
This application is important because it saves time and labor, both of which are typically in short supply. When a pharmaceutical company targets a protein, there are many more potential ligands to consider than could be examined enumeratively. Each possible ligand may have several rotatable bonds, offering a multidimensional optimization problem of finding the best overall conformation. Rather than trying to look at each rotation individually, an evolutionary algorithm can quickly discover which ligands best fit the target protein's binding site and therefore have potential as candidate drugs. Chemists can then focus their attention on these candidates. The result



[5] To test its applicability to drug design, an evolutionary algorithm was used to select candidate molecules, or ligands, that would dock to the receptor site of an HIV-1 protease protein [shown, far left, top, as a biochemist's ribbon diagram]. The best 150 out of 1000 possible solutions in the 2nd generation look messy [near left, top]. But the algorithm makes progress in the 70th [far left, middle], 92nd [near left, middle], and 149th [far left, bottom] generations.

The best structure obtained [green, near left, bottom] closely matches one found experimentally [white], indicating the usefulness of evolutionary techniques.

(Images from "Molecular recognition of the inhibitor AG-1343 by HIV-1 protease: Conformationally flexible docking by evolutionary programming," by Daniel K. Gehlhaar et al., *Chemistry & Biology*, Vol. 2, 1995.)



Decepticon12345> hi
David1101> hi
Decepticon12345> gm
David1101> thanks

[6] In a recent game of checkers played on-line, David1101's moves were secretly determined by an evolved neural network, whereas Decepticon12345 was a human competitor with an expert rating.

At the point in the game shown, the neural network, playing white, has just earned a king and the human expert has acknowledged the move by typing "gm," meaning "good move."

is an improved, more cost-effective procedure for screening drug leads.

Other real applications of evolutionary algorithms are being made in scheduling, supply-chain management, and medical diagnosis. They may even be used as the basis for combat simulations to train military personnel; an evolutionary algorithm can serve as an interactive adversary that learns and adapts its tactics as the strategic situation changes. Moreover, evolution is being used as a principle for creating new hardware designs, in terms of both physical devices and electronic circuits.

In the longer run, evolutionary algorithms may play a pivotal role in devising truly intelligent machines—computers that can learn on their own. The real challenge is not so much to make a computer that can compete with people; in many narrowly focused applications, such as games, this is easily accomplished by translating human expertise into programmed rules of behavior. The real objective is to give the computer a way to learn how to behave like an expert *without* relying on human expertise.

A recent step in this direction is an evolutionary algorithm that learned to play checkers at a very high level of competency without relying on any techniques associated with human expertise [see the paper by Chellapilla and Fogel in To Probe Further, this page]. The algorithm used a population of artificial neural networks (computational models based loosely on the architecture of a brain) to measure the quality of any checkerboard position that would be encountered in the game. Each neural network was in essence a strategy for playing, because it generated a score indicating which positions would be favored over others. The evolutionary algorithm pitted these neural networks against each other, starting from a completely random beginning and applying variation and selection iteratively.

After only 10 generations, the best network in the population was able to defeat its inventors easily (admittedly, Chellapilla and I are poor players). After 100 generations, the best-evolved network played against people at the Microsoft Network Gaming Zone Web site (www.zone.com) and, based on the number of games played and the network's performance, earned a "Class A" rating—one level below the designation of "expert" following the United States Chess Federation rating system. Of course, opponents were not told that they were playing against a program, nor did any of them guess that their rival was a program. In fact, some people praised the apparent ingenuity of their competitor [Fig. 6].

The crucial achievement of this work is that the neural networks did not assess their position by using human expertise—piece mobility, control of the center of the board, having a path that can earn a king, or any other sophisticated technique a human checkers player would use. Instead, they used only the actual positions of the pieces on the board and the piece differential, that is, the number of pieces ahead or behind. The evolutionary algorithm extracted all it needed to know about how to play at a near-expert level simply by playing the game and receiving feedback on the ultimate outcome of each game. Whatever it learned, it certainly did not learn from its creators who, as noted above, know very little about how to play a good game of checkers. Experiments are continuing in an effort to achieve an expert-level rating, which would place the evolved strategy in the top 1 percent of all players who have registered at the Internet gaming site.

The ability for a computer to gain proficiency at a game of skill such as checkers simply by teaching itself opens up a far greater possibility: having computers learn about other facets of the real world without relying on people to program in all the requisite knowledge. As the cost of computing keeps declining, the amenability of evolutionary algorithms to solving real-world problems will soar. This is particularly so because of the parallel nature of evolutionary approaches. With natural evolution, individuals are always being evaluated in parallel by their environment. So by relying on a cluster of computers (a so-called "pile of PCs"), a practitioner can take less time to solve difficult problems; evaluate individual solutions in parallel rather than in series; and have them migrate from computer to computer. The very large populations this protocol allows yield better solutions faster than do small ones.

The products that will emerge from these massively parallel designs remain a matter of speculation. Some even foresee the use of evolutionary algorithms to design surrogate artificial brains that will supplant human cognition, as did Ray Kurzweil in *The Age of Spiritual Computing* (Viking, 1999). Whether or not this conjecture becomes reality, there is no doubt that evolutionary algorithms will become a mainstay of problem solving in the coming years. ◆

TO PROBE FURTHER

An introduction to the art of designing and applying evolutionary algorithms to real-world problems is provided in *How to Solve It: Modern Heuristics* by Zbigniew Michalewicz and David B. Fogel (Springer, 2000). The book also details classic optimization techniques.

Recent advances in evolutionary computation are found in many conferences and symposia internationally, including the annual Congress on Evolutionary Computation, co-sponsored by the IEEE Neural Networks Council, the IEE, and the Evolutionary Programming Society. Information on the upcoming conference can be found at http://pcgipseca.cee.hw.ac.uk/cec2000/

"Evolution, neural networks, games, and intelligence" are discussed in the paper of that title by Kumar Chellapilla and David B. Fogel in the *Proceedings of the IEEE*, Vol. 87, no. 9, pp. 1471–96, September, 1999.

The relationship between evolutionary algorithms and machine intelligence is explored in *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* by D.B. Fogel (2nd edition, IEEE Press, Piscataway, N.J., 2000). Many of the foundational papers in the field, as far back as 1956, are reprinted in *Evolutionary Computation: The Fossil Record* (D.B. Fogel, editor, IEEE Press, 1998).

ABOUT THE AUTHOR

David B. Fogel is executive vice president and chief scientist of Natural Selection Inc., located in La Jolla, Calif. He is the founding editor-in-chief of the *IEEE Transactions on Evolutionary Computation* and the author of more than 180 publications in the field, including four books. He was elected a Fellow of the IEEE in 1999.

Spectrum editor: Richard Comerford